Operating System/2™ Programming Tools and Information Version 1.2

Dialog Tag Language Guide and Reference

64F3829

Operating System/2™
Programming Tools
and Information
Version 1.2

Dialog Tag Language Guide and Reference

First Edition (September 1989)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Note to US Government users - Documentation related to Restricted Rights - Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

© Copyright International Business Machines Corporation 1989. All rights reserved.

About This Book

This book describes the Dialog Tag Language (DTL), the tag-based markup language used for creating displayable panels, messages, key lists, and other elements of Dialog Manager applications. The Dialog Manager uses these elements in dialog applications that interact with end users.

This book should be used with the *Dialog Manager Guide and Reference*, which describes the services and conventions for the OS/2® Dialog Manager.

Who This Book Is For

The intended audience for this book is application developers, although technical writers and others involved in designing and creating dialog applications for Dialog Manager will also find this book useful. This book assumes you are already familiar with designing DM applications and have knowledge of fundamental programming concepts.

This book is written for the application developer, not the end-user. Therefore, when we address *you* in the text, we are referring to the intended audience; the application developer. When we talk about *users*, we are referring to the end-users of the applications you design.

How This Book Is Organized

The information in this book is presented in two parts.

Part 1, "Guide to the Dialog Tag Language"
This section guides you through the steps involved in using the Dialog Tag Language for designing, building, and formatting dialog elements for Dialog Manager applications. It shows you how to create application panels and help panels, messages, command tables, and other dialog elements by providing examples of actual source file markup and the formatted results.

Part 2, "Dialog Tag Language Reference"
This section provides an alphabetic reference of the Dialog Tag Language. Included are references for the DTL commands, markup declarations, and tags. Each reference contains a diagram and explanation of the syntax, a description, and an example of how to use them.

The windows illustrated in this book are intended to show examples of panels defined using the Dialog Tag Language. They are not intended as exact representations. The actual window may differ slightly when you display the panels you define using the Dialog Tag Language.

Refer to the *OS/2 Getting Started* book for information about how to name files. Also refer to the "Technical Bulletin" section of the *OS/2 Product Information* book for information regarding an OS/2 Standard Edition Version 1.2 update. This update will contain additional rules for naming files.

OS/2 is a registered trademark of the International Business Machines Corporation.

Related Publications

The following books contain information that may be useful when you are planning and writing DM applications:

- *Dialog Manager Guide and Reference* provides a complete description of the Dialog Manager services and syntax, and how to use it to create dialog applications.
- *Dialog Manager and Dialog Tag Language Reference Summary* provides a quick reference for the syntax of Dialog Manager services and Dialog Tag Language tags.
- *Programming Guide* provides a complete description of how to use the Presentation Manager to create PM applications.
- *Presentation Manager Programming Reference* describes the Presentation Manager calls with which your DM application can interact.
- *Programming Overview* defines programming concepts needed to write OS/2 applications.
- *Systems Application Architecture Common User Access Advanced Interface Design Guide*, SC26-4582, describes the advanced interface for software applications so that you can create consistent Operating System/2 (OS/2) applications.
- *Systems Application Architecture Common Programming Interface Presentation Reference*, SC26-4359, describes the interface and protocol on which the OS/2 Presentation Manager is based.
- *Systems Application Architecture Common Programming Interface Dialog Reference*, SC26-4356, defines the interface and protocol on which the OS/2 Dialog Manager is based.

Contents

Part 1. Guide to the Dialog Tag Language

About Part 1

Chapter 1. Introduction to the Dialog Tag Language 1-1
Why the Dialog Tag Language (DTL)? 1-1
What Is the Dialog Tag Language? 1-2
Dialog Elements 1-3
Application Panels 1-4
Help Panels 1-5
Messages 1-7
Application Command Table 1-7
Key Mapping Lists 1-8
Variables and Variable Classes 1-8

Chapter 2. How to Use the Dialog Tag Language 2-1
Syntax Conventions 2-1
Attributes and Values 2-1
Tag Text 2-2
Nesting tags 2-3
Markup Declarations 2-4
Declaring the Document Type 2-4
Including Comments in Your Markup 2-5
Entities and Parameter Entities 2-6
Predefined Entities 2-12

Chapter 3. Getting Started: Designing Application Panels 3-1
Defining Application Panels: The PANEL Tag 3-1
The Window Title 3-2
Window Size (Width and Depth) 3-2
Message Type 3-3
Key Mapping Lists 3-3
Associated Help Panels 3-4
Cursor Placement 3-4
Defining Action Bars and Pull-Downs 3-6
Pull-Down Choice Actions 3-9
Mnemonics 3-9
Action Bar Help 3-10
Help Action Bar Pull-down 3-11
Unavailable Pull-Down Choices 3-12
Check Marks for Pull-Down Choices 3-14
Defining a Scrollable Area: The AREA Tag 3-15
Defining Top and Bottom Instructions 3-19
Defining Panel Layout: The DIVIDER and REGION Tags 3-20
The DIVIDER Tag 3-20
The REGION Tag 3-21
Defining a Command Area 3-25

Chapter 4. Variables and Variable Classes 4-1
Declaring Variables 4-1
Defining Variable Classes 4-2

Variable Class Types 4-3
Variable Validation 4-9
Translate Lists 4-10
Validity Checks 4-15
Overriding Variable Classes 4-17
Variable Class Display Length and Field Size 4-18

Chapter 5. Application Panel Fields 5-1
Field Prompts 5-1
Defining Data Fields 5-5
Data Field Width 5-7
Data Field Descriptions 5-10
Data Field Help 5-11
Other Data Field Attributes 5-12
Defining Selection Fields 5-13
Single-choice Fields 5-14
Multiple-choice Fields 5-15
Immediate-action Fields 5-17
Selection Field Choice Actions 5-18
Mnemonics 5-19
Selection Field Help 5-19
Unavailable Choices 5-20
Selection Width 5-23
Data Columns 5-24
Defining Selection Lists 5-26
Other Selection List Attributes 5-29
Selection List Help 5-30
Defining List Fields 5-30
List Column Width 5-32
Rows in a List Field 5-33
List Field Help 5-33
List Field Modification 5-33
List Field Grids 5-34
Other List Column Attributes 5-34
Sizing the Selection List and List Field 5-36

Chapter 6. Information Regions 6-1
Defining Basic Text 6-2
Paragraphs 6-2
Headings 6-4
Lines 6-5
Examples 6-6
Figures 6-7
The Figure Caption (FIGCAP) Tag 6-10
Defining Lists 6-11
Simple Lists 6-11
Unordered Lists 6-13
Definition Lists 6-18
Parameter Lists 6-22
Nesting Tags within Lists 6-23
Nesting Lists within Lists 6-25
Alerting Users: Notes, Warnings, and Cautions 6-26
Notes 6-27
Warnings 6-29
Cautions 6-30
Using Information Regions with Other Panel Elements 6-31

Chapter 7. Help Panels 7-1
Defining Help Panels 7-1
Help Panel Information Regions 7-2
Defining Help Panel Text 7-3
Defining Reference Phrases 7-8
Defining Help Index Topics and Synonyms 7-10
Defining Command Help 7-13
Providing Help for Application Panels 7-14
Keys Help 7-15

Chapter 8. Messages 8-1
Defining Messages 8-1
Specifying Message Types 8-2
Assigning Messages 8-3
Variable Substitution 8-4
Substituting a Variable Value 8-5
Substituting a Field Value 8-5

Chapter 9. The Application Command Table 9-1
Defining the Application Command Table 9-1
Specifying Command Actions 9-4
Truncating Commands 9-7

Chapter 10. Defining Key Mapping Lists 10-1
Assigning Keys and Actions 10-1
Default Key Assignments 10-2
Combining Key Assignments 10-2
Displaying Keys 10-3

Chapter 11. DTL Shortcuts 11-1
Defining Panel Defaults 11-1
Reducing Tag Markup 11-3
Omitted Optional End Tags 11-4
Empty Start Tags 11-4
Empty End Tags 11-5
Unclosed Start Tags 11-6
Unclosed End Tags 11-6
Null End Tags 11-6
Comprehensive Examples 11-7

Chapter 12. Compiling and Displaying Dialog Elements 12-1
General Rules for DTL Commands 12-1
Compiling and Verifying Source Files 12-2
Verifying Markup Syntax 12-2
DTLC Messages 12-2
Compiling Source Files 12-3
Online Description 12-6
Deleting Dialog Elements 12-6
Online Description 12-7
The DTL Display Utility 12-7
Accessing the Display Utility 12-8
Displaying Panels and Messages 12-8
Restrictions on the Display Utility 12-9

Part 2. Dialog Tag Language Reference

About Part 2
How to Read the Syntax Diagrams

Chapter 13. DTL Command Reference 13-1
DTLC 13-2
DTLD 13-10

Chapter 14. Markup Declaration Reference 14-1
Document Type Declaration 14-2
Entity Declarations 14-3

Chapter 15. Tag Reference 15-1
How to Use the Nested Tag Tables 15-2
AB (Action Bar) 15-3
ABC (Action Bar Choice) 15-5
ACTION (Action) 15-7
AREA (Area) 15-11
ASSIGNI (Assignment List Item) 15-14
ASSIGNL (Assignment List) 15-16
BOTINST (Bottom Instruction) 15-18
CAUTION (Caution) 15-20
CHECKI (Validity Check Item) 15-22
CHECKL (Validity Check List) 15-26
CHOICE (Selection Choice) 15-28
CMD (Command Definition) 15-34
CMDACT (Command Action) 15-37
CMDAREA (Command Area) 15-40
CMDTBL (Command Table) 15-42
DD (Definition Description) 15-44
DDHD (Definition Description Header) 15-46
DIVIDER (Area Divider) 15-48
DL (Definition List) 15-50
DT (Definition Term) 15-54
DTACOL (Data Column) 15-56
DTAFLD (Data Field) 15-60
DTAFLDD (Data Field Description) 15-66
DTHD (Definition Term Header) 15-68
FIG (Figure) 15-70
FIGCAP (Figure Caption) 15-73
HELP (Help Panel) 15-75
Hn (Heading) 15-77
ICMD (Index command) 15-79
ICON (Icon Resource) 15-81
INFO (Information Region) 15-82
ISYN (Index synonym) 15-84
ITOP (Index Topic) 15-86
KEYI (Key Item) 15-88
KEYL (Key Mapping List) 15-91
LI (List Item) 15-92
LINES (Lines) 15-95
LIT (Literal) 15-97
LP (List Part) 15-99
LSTCOL (List Column) 15-101
LSTFLD (List Field) 15-106
M (Mnemonic) 15-110
MSG (Message) 15-114

MSGMBR (Message Member) 15-116
NOTE (Note) 15-119
NT (Note) 15-121
OL (Ordered List) 15-123
P (Paragraph) 15-125
PANDEF (Panel Default) 15-129
PANEL (Panel) 15-132
PARML (Parameter List) 15-137
PD (Parameter Description) 15-140
PDC (Pull-Down Choice) 15-142
PDSEP (Pull-Down Separator) 15-145
PT (Parameter Term) 15-147
REGION (Region) 15-149
RP (Reference Phrase) 15-152
SELCOL (Selection Column) 15-154
SELFLD (Selection Field) 15-156
SELLST (Selection List) 15-161
SL (Simple List) 15-165
SORTKEY (Sort Key) 15-167
T (Truncation) 15-169
TOPINST (Top Instruction) 15-171
UC (User Control) 15-173
UL (Unordered List) 15-176
VARCLASS (Variable Class) 15-178
VARDCL (Variable Declaration) 15-186
VARLIST (Variable List) 15-188
VARSUB (Variable Substitution) 15-190
WARNING (Warning) 15-192
XLATI (Translate Item) 15-194
XLATL (Translate List) 15-197
XMP (Example) 15-202

Appendix A. Nesting Order of Dialog Tag Language Tags A-1

Glossary X-1

Index X-7

Part 1. Guide to the Dialog Tag Language

About Part 1

This section contains the following chapters:

Chapter 1, "Introduction to the Dialog Tag Language"
This chapter contains an introduction to the Dialog Tag Language, and descriptions of the dialog elements you define with the Dialog Tag Language.

Chapter 2, "How to Use the Dialog Tag Language"
This chapter explains the syntax conventions of the Dialog Tag Language.

Chapter 3, "Getting Started: Designing Application Panels"
This chapter tells you how to design application panels.

Chapter 4, "Variables and Variable Classes"
This chapter tells you how to declare variables, define variable classes, and define variable validation.

Chapter 5, "Application Panel Fields"
This chapter tells you how to define interactive fields for application panels.

Chapter 6, "Information Regions"
This chapter tells you how to define information regions on application panels.

Chapter 7, "Help Panels"
This chapter tells you how to define help panels.

Chapter 8, "Messages"
This chapter tells you how to define messages.

Chapter 9, "The Application Command Table"
This chapter tells you how to define application commands and the application command table.

Chapter 10, "Defining Key Mapping Lists"
This chapter tells you how to define key mapping lists.

Chapter 11, "DTL Shortcuts"
This chapter provides techniques on saving steps in DTL markup.

Chapter 12, "Compiling and Displaying Dialog Elements"
This chapter describes the steps involved in compiling and storing dialog elements. A section on displaying panels and messages is also provided.

Chapter 1. Introduction to the Dialog Tag Language

This book introduces you to the Dialog Tag Language (DTL), a tag-based language used to define many of the elements that make up the type of application known as a *dialog*, the communication between a person and a computer. We will tell you how to define these elements using DTL and how to prepare them for the Dialog Manager.

The elements you produce with DTL are used by the Dialog Manager as the user interface to your DM applications. The programs you write using Dialog Manager and a programming language control the display and function of the dialog elements you create for an application. For a complete description of the Dialog Manager, refer to the *Dialog Manager Guide and Reference*.

This chapter provides an overview of DTL, and the dialog elements you create with DTL for your DM applications. We introduce you to DTL by discussing the following main topics:

- Why DTL?

 This section talks about why you would want to use DTL to create elements for DM applications.

- What is the Dialog Tag Language?

 This section explains what the Dialog Tag Language is and how it works.

- Dialog elements

 This section provides descriptions and examples of the dialog elements. These elements are:

 - Application panels
 - Help panels
 - Messages
 - An application command table
 - Key mapping lists.

- Variables and variable classes.

 The Dialog Tag Language provides you with the ability to define and declare *variables* and *variable classes* for your applications. You'll find a brief description of variables and variable classes at the end of this chapter.

Why the Dialog Tag Language (DTL)?

If you are already familiar with a tag-based markup language, such as IBM's Publishing Systems' BookMaster, you will find that DTL is very similar. We created DTL for many of the same reasons that we created BookMaster. For example:

- Markup tags are easy to use. Since they are short and relate directly to the structure of the dialog elements, they are also easy to remember.
- DTL lends flexibility to application development. Panels can be quickly changed without your having to tediously line up titles and fields. This gives you greater control over application development and updates.
- DTL provides consistency when many programmers are working on the same application, or when programmers who are new to your company must update

existing applications. Since each programmer is using the same tags, only minor adjustments may be needed to achieve complete uniformity.

- DTL techniques improve the way in which interactive programs, like DM applications, are developed. The language concentrates on the role of the various elements and their interrelationships, as well as on defining their form and appearance.
- DTL also enforces some of the guidelines defined by the Systems Application Architecture Common User Access (CUA), and enables many more, so you do not have to be familiar with all of the CUA guidelines. Therefore, the CUA skills required by programmers who are developing CUA-conforming applications are significantly reduced.
- DTL enables National Language Support (NLS). Special translation options allow you to define unique language formatting requirements for your DM applications.
- Most of the elements you define with DTL for a DM application can be easily migrated to other SAA environments. This portability of function is an important feature.

In other words, if you are looking for an application development and maintenance system that is sophisticated, flexible, and easy to use, that's DTL.

What Is the Dialog Tag Language?

In "Why the Dialog Tag Language (DTL)?" on page 1-1, we referred to DTL as a tag-based markup language that is similar to IBM Publishing Systems' BookMaster. The two have much in common. For example, *markup* is a term that is usually associated with documentation. It is an old typesetter's term that formerly meant the instructions with which a document was "marked up" to show how the document should be set in type.

Today, this definition has been expanded to include information that is added to a document to enable a person or system to process it. Just as markup information can describe a document's characteristics or the processing to be performed, it can also describe the characteristics or processing related to dialog elements. This is where the tags come in.

We call DTL a *tag-based* markup language simply because the markup consists of tags that determine not only what each element is, but also how it is processed. To convert the dialog elements into a format that is usable by Dialog Manager, you must process them with the DTL compiler.

Another thing that DTL and BookMaster have in common, of course, is the tags themselves, which have the following similarities:

- They are very short and easy to remember.
- They are often accompanied by text.
- Many DTL tags are almost identical to corresponding BookMaster tags.

These are all reasons that familiarity with BookMaster will make it easy to learn DTL. For example, we created the preceding bulleted, or *unordered* list by editing a file using an ASCII editor and typing the following:

```
:ul compact.
  :li.They are very short and easy to remember.
  :li.They are often accompanied by text.
  :li.Many DTL tags are almost identical to corresponding BookMaster tags.
:eul.
```

You could create a similar list for an information panel by using the following DTL tags with the same text:

```
<ul compact>
  <li>They are very short and easy to remember.
  <li>They are often accompanied by text.
  <li>Many DTL tags are almost identical to corresponding BookMaster
      tags.
</ul>
```

Here, the and tags, respectively, begin and end the unordered list. This type of list is called an unordered list because the list items are not numbered. The individual list items are defined by the tags and consist of the accompanying text.

As you can see from the preceding example, DTL tags act as control words that specify how the text of source files is interpreted by the DTL compiler. This concept is based on the Standard Generalized Markup Language (SGML), which is a standard of the International Standards Organization (ISO). The conventions of the Dialog Tag Language are based on the SGML standard.

After you are finished marking up a source file, you use the compiler to convert the file into a format usable by your DM application. In addition to processing the file, the compiler also checks and verifies the syntax of your markup, and notifies you of any errors. After compilation, the elements you defined in your source file are stored within libraries.

DM also provides a display utility that you can use to display application panels and messages on your screen after they have been compiled. Displaying your panels is a good idea to make sure that they format properly.

You should now have a basic understanding of DTL and how it works. The next section will build on this understanding by describing the types of elements that you can define with DTL.

Dialog Elements

This section provides a descriptive overview of the dialog elements you can create for your DM applications.

These elements include:

- Application panels
- Help panels
- Messages
- An application command table
- Key mapping lists.

Application panels, help panels, and messages are the displayable elements users interact with. They appear within Presentation Manager primary and pop-up windows on the user's display device. The remaining elements, while transparent

to the user, provide the important functions of defining and storing the various types of application data.

Application Panels

Application panels are the primary element of the user interface for an application. They allow users to interact with Dialog Manager applications through the use of data fields, selection fields, and other interactive fields. These fields are used to display lists of choices and to allow the user to enter data. Application panels appear in primary and pop up windows.

Figure 1-1 shows an application panel within a primary window. Following that is a list that defines each of the elements that make up an application panel.

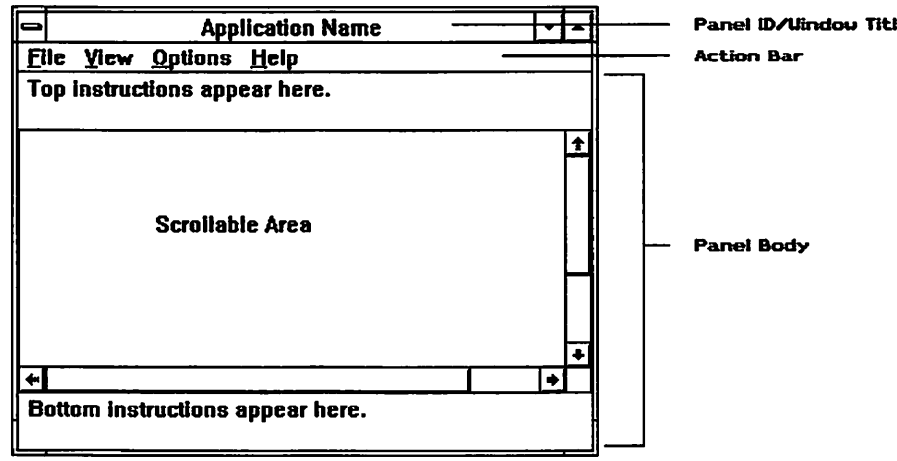


Figure 1-1. Application Panel Layout

Application Panel Elements

Panel ID/Window title

The window title appears in the title bar at the top of the window. The panel ID, when visible, appears on the left side of the title bar separated from the window title text by a dash (—). The user can choose to display the panel with or without the panel ID.

Action bar

The action bar appears directly below the window title bar. It contains keyword choices that provide users access to available actions that apply to the current panel. When the user selects an action bar choice, a pull-down menu containing additional choices appears directly below the action bar choice.

Panel body

The panel body serves as the main work area of the panel. The panel body contains the input and output fields, selection fields, and other text the user interacts with.

Additionally, the panel body can contain optional top and bottom instructions, which provide instructional text to the user. Top instructions appear immediately below the action bar and above the interactive fields on the panel. Top instructions tell the user how to interact with the panel. Bottom instructions appear immediately below the interactive fields on the panel. Bottom instructions tell the user how to interact with the panel, or how to continue with the application.

You can define the area of the panel body that contains the interactive fields (between the top and bottom instructions) as being *scrollable*, or containing information that is not visible to the user on the initial display of the panel in the window. In this case, a vertical scroll bar, a horizontal bar, or both appear as needed to allow the user to scroll the unseen portion of the window into view. These scroll bars do not appear if all of the contents of the panel body fit within the current window dimensions.

In addition to the preceding list of panel elements, you can also use DTL to define *command areas* and *function key areas* for application panels that are intended for migration to host environments.

The optional command area, which appears at the bottom of the panel body, consists of two components: the command field prompt and the command entry field. Application users use the command entry field to enter commands or requests to the DM application.

The optional function key area, which appears at the bottom of the panel immediately below the command area (if one is defined), contains the key assignments for dialog actions valid for the application panel.

Chapter 3, "Getting Started: Designing Application Panels" on page 3-1 tells you how to define application panels and panel elements.

Help Panels

Help panels appear in response to user requests for assistance during DM application sessions. The Dialog Manager processes these requests and displays the help panels in help windows adjacent to the primary application window.

Using DTL, you can create help panels that provide help for:

- An entire application panel
- A specific field on an application panel
- Messages
- Action bar choices and pull-down choices
- Commands
- One or more words on another help panel (reference phrases).

Figure 1-2 on page 1-6 illustrates a help panel. Following that is a list that defines each of the elements that make up a help panel.

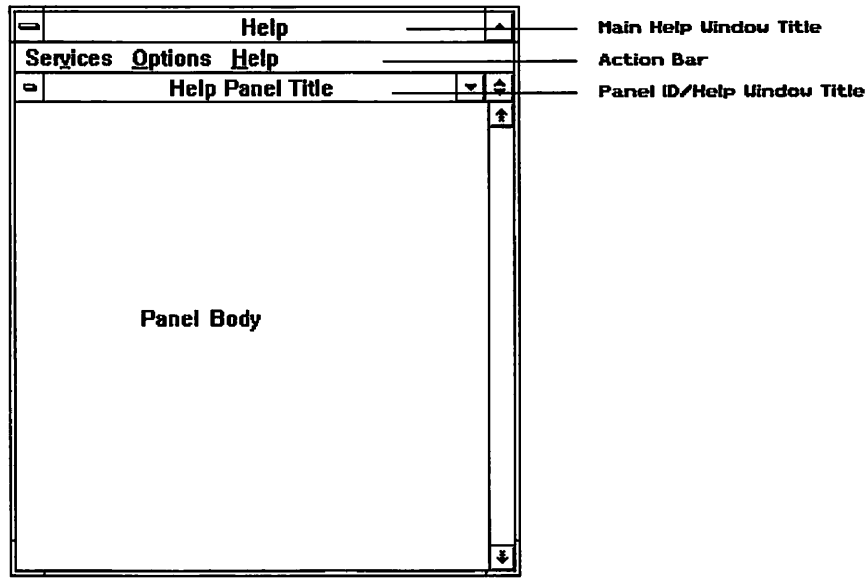


Figure 1-2. Help Panel Layout

Help Panel Elements

Main help window title

The main help window title appears at the topmost portion of the window. The window title text is horizontally centered in the title bar.

Action bar

The action bar appears directly below the main help window title bar. It contains automatically-supplied action bar choices. When the user selects a choice, a pull-down menu containing additional choices appears below the action bar choice.

Panel ID/Help text window title

The window title appears in the title bar at the top of the help text window. The panel ID, when visible, appears on the left side of the title bar separated from the window title text by a dash (−).

Help text window

The help text window contains the body of help panel text.

Text within the help panel is protected, which means that the user cannot interact with the text except to request help through designated *reference phrases*.

The panel body of a help panel is always scrollable. A vertical scroll bar is always displayed, but is only active when the text defined for the panel does not fit within the window. Additionally, if the user sizes the help text window, the text of the help panel is automatically reformatted.

In addition to help panels, you can define entries for the help index that provides a list of all of the help topics in an application. The user can search for specific help topics using the provided Search Index function. The user can select any help panel in the application from the help index if an index entry for the panel has been specified. The help index also contains system-defined help topics.

The table of contents contains the entries for the titles of all of the help panels in your application. The user can view the table of contents in the provided Contents window and choose a help topic. In addition to the panel titles, the table of contents contains system-defined help topics.

Chapter 7, "Help Panels" on page 7-1 tells you how to define help panels and help indexes.

Messages

You can use DTL to define messages that display in message pop-ups in response to a user request or action. Messages can confirm a user-requested action, report an error in user input, or notify the user of an error or exception condition. Figure 1-3 illustrates a message pop-up.

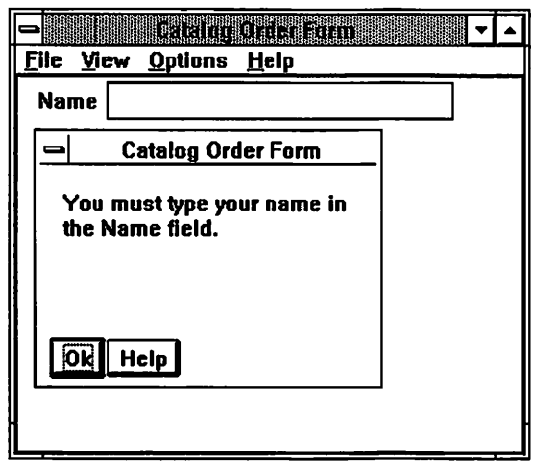


Figure 1-3. Message Pop-Up

The messages you define for an application are stored within *message members*. You use DTL to define the messages and message members.

Chapter 8, "Messages" on page 8-1 provides a complete description of defining messages and message members.

Application Command Table

You can use DTL to define commands that perform actions requested by the user. The valid commands for an application are defined and stored within an internal application command table. You can define only one command table for an application.

Valid commands include those assigned to pull-down choices, selection field choices, function keys, and commands entered in command entry fields.

Chapter 9, "The Application Command Table" on page 9-1 tells you how to define commands and application command tables.

Key Mapping Lists

The key assignments that are active for an application are defined and stored within key mapping lists. These key assignments allow the user to request commands and other actions through the use of function keys. You can choose to display the valid key assignments for your application in the function key area of application panels.

Chapter 10, "Defining Key Mapping Lists" on page 10-1 tells you how to define key assignments and key mapping lists.

Variables and Variable Classes

Variables are used to communicate information from an application to DM and to communicate information from DM back to the application. You use DTL to define the variables and variable classes for an application.

Each variable you define for an application must be *declared*, or identified within a variable list. In addition, each variable must be associated with a variable class that defines its type and length characteristics. The variable class can also be used to define translations and validity checks that are used when a value is displayed on a panel or entered by a user.

Chapter 4, "Variables and Variable Classes" on page 4-1 tells you how to declare variables and define variable classes.

Chapter 2. How to Use the Dialog Tag Language

This chapter describes the syntax conventions of the Dialog Tag Language and tells you how to use it to create dialog element source files for your Dialog Manager applications.

Also included in this chapter is a section on *markup declarations*. Markup declarations are control statements that you can use in your source files in addition to tags.

Syntax Conventions

The markup style of the Dialog Tag Language is based on the International Standards Organization (ISO) Standard Generalized Markup Language (SGML). Markup languages allow you to specify, through the use of *tags*, how the text of a file is to be formatted when you compile it for use by an application. Because the Dialog Tag Language is a markup language, you must follow special rules and conventions when using it to define elements.

The DTL tags act as control words that determine how the text in the source files you create for Dialog Manager applications is used. Each tag or control word, is enclosed within a set of delimiter symbols that distinguish the tag as a control word (as opposed to general text). *Start* tags, which initiate text interpretation, are preceded by the start tag open delimiter (<) and followed by the close delimiter (>). *End* tags, which explicitly end text interpretation, are preceded by the end tag open delimiter (</) and followed by the close delimiter (>).

For example, the DTL tags used for defining the beginning and end of an application panel are the PANEL tag and its matching end tag, which look like this:[1]

```
<panel>
</panel>
```

Attributes and Values

Many DTL start tags contain *attributes* and *values* that define various physical and operating characteristics of the dialog elements. While most attributes and values are optional, or contain default settings, some are required.

For example, the PANEL tag has a required NAME attribute that must be specified in order for the compiler to identify the panel. The value you assign to the NAME attribute must be unique for each panel in a source file. This PANEL tag has the NAME value "panel1":

```
<panel name=panel1>
</panel>
```

The PANEL tag also has two optional attributes, DEPTH and WIDTH, whose values specify the dimensions of the window the panel appears in. For these types of attributes, you specify a numeric value.

[1] End tags are not required for all of the DTL tags. In many cases, the tag is implicitly ended by other start tags. For this reason, we don't use optional end tags in the markup examples in this book. Chapter 15, "Tag Reference" on page 15-1 contains a detailed description of each of the DTL tags, including which tags require an end tag.

```
<panel name=panel1 depth=20 width=40>
</panel>
```

Values for some of the tag attributes are predetermined; that is, you can choose from one of a number of keyword values for the tag. For example, the FIG (figure) tag has a FRAME attribute that specifies the top and bottom borders of the figure. The value you assign to the FRAME attribute can be either RULE, which produces a visible border above and below the figure, or NONE, which results in a figure without a border. No other value is acceptable for the FRAME attribute.

RULE is the default value, which means that the figure formats with visible borders if you do not specify the FRAME attribute.

The markup for a figure without ruled borders looks like this:

```
<fig frame=none>
</fig>
```

When coding values for attributes, you must use single or double quotes to enclose values that contain characters other than A–Z, 0–9, a hyphen (–), or a period (.).

For example, the value assigned to the TYPE attribute of this VARCLASS tag contains a blank, so the value must be enclosed in quotes:

```
<varclass name=boolean type='char 1'>
```

Some attributes can be assigned either a specific value, such as a number or a character string, or a variable name. To distinguish a variable name from a specific value, you must enclose the variable name in quotes and precede it with a percent (%) sign. This convention is called *% notation*. The percent sign distinguishes the variable name from a specific value.

The TOPROW attribute in the following example uses % notation to specify a variable named "listvar":

```
<lstfld toprow='%listvar'>
```

The length of any attribute value is limited to 253 characters. This includes the lengths of any entity references that are a part of the value. Any additional length restrictions are described in Chapter 15, "Tag Reference" on page 15-1.

Generally, you can code tags, attributes, and values in uppercase, lowercase, or mixed case; the results are always the same regardless of case. The conventions you must follow for case-sensitive processing for each tag are described in Chapter 15, "Tag Reference" on page 15-1.

Tag Text

The content, or text of a tag is coded immediately following the start tag. This is the actual text of the tag that is subject to formatting and translation. The text is processed according to the type of tag it follows.

For example, the text following this P (paragraph) tag is the actual text that appears in the panel after formatting:

```
<p>The copy command allows you to copy single or multiple forms.
```

Because the tag text is processed according to the tag characteristics, not the way it is written in the source file, the paragraph could also be marked up using more than one line, like this:

```
<p>
The copy command allows you to
copy single or multiple forms.
```

The formatted result is the same in either case.

In most cases, there is no limit to the amount of text you can code as tag content. However, keep in mind that the text of some tags, such as the title of a PANEL tag, should be limited because of size constraints on the panel they are coded within. Chapter 15, "Tag Reference" on page 15-1 describes any text restrictions (if they exist) for each of the tags.

In most cases, multiple lines of text are concatenated. Concatenation, leading blanks, and trailing blanks are processed as follows:

- Blanks following start tags are preserved except in tag text that does not permit blanks. For instance, blanks following a CMD tag are not preserved.
- Leading and trailing blanks between lines of text are not preserved. Instead, they are compressed to a single blank when the lines are concatenated.[2]
- The first line of tag text may start on the same line as the start tag, or on the next line. The formatted result is the same.

The text of some tags, such as the FIG, LINES, and XMP tag, allow you to control where lines break. That is, within the range of the tag, each output line is ended at the same point that you ended the input line. With these tags, multiple lines are not concatenated, and all blanks are preserved.

Nesting tags

It is often necessary to code certain tags (and their text) within other tags (between the start and end tags). This is called *nesting*.

A good example of nesting is the relationship between the DL (definition list) tag, the DT (definition term) tag, and the DD (definition description) tag. The DL tag specifies a definition list and the DT and DD tags specify the terms and descriptions of the items within the definition list. Consequently, the DT and DD tags must be nested within a DL tag and its matching end tag if the list is to format properly.

Here is an example:[3]

```
<dl>
  <dt>This is a definition term.
  <dd>This is a definition description.
  <dt>Another term.
  <dd>Another description.
</dl>
```

[2] Concatenation of lines for DBCS languages is handled in a manner appropriate for the specific language.

[3] Although it isn't required, we indented the nested tags in this example to illustrate nesting levels. You can also do this in your own source files.

There are several tags that must be nested within the actual text of another start tag. These tags serve to modify or highlight some aspect of the text.

In this example, the nested CHOICE tags follow the SELFLD start tag and precedes the SELFLD end tag. The M (mnemonic) tags nest within the text of the CHOICE tags.

```
<panel name=panel1>
  <area>
    <selfld>
      <choice>Order <m>One
      <choice>Order <m>Two
    </selfld>
  </area>
</panel>
```

Nesting tags can take on many different forms and can be complex. For example, some tags allow multiple tags or multiple occurrences of the same tag to be nested, while other tags do not allow nesting of any tags. You can also nest levels of certain tags, that is, nested tags within other nested tags. Additionally, in many instances, you must nest certain tags within other tags. The tag descriptions in Chapter 15, "Tag Reference" on page 15-1 describe the allowed and required conditions for nesting each of the DTL tags.

Markup Declarations

Besides tag markup, you can also include *markup declarations* in your source files to define other, related information. Markup declarations are control statements that specify how other markup (such as tags) within a source file is to be interpreted.

For example, in order for the compiler to recognize your source files as being intended for DTL compilation, you must include a *document type* declaration at the beginning of the source file.

Like tags, markup declarations must be enclosed within a set of delimiter symbols so the compiler can distinguish the declaration as a control statement. All markup declarations are preceded by the <! symbol and followed by the > symbol.

DTL supports three types of markup declarations:

- Document type declarations
- Comments
- Entity declarations.

Declaring the Document Type

Before you can compile a source file that contains the tag markup for dialog elements, you must declare the *document type*. Do this by coding the DOCTYPE declaration at the beginning of the source file. The DOCTYPE declaration looks like this:

```
<!doctype dm system>
```

Where:

<! Begins the markup declaration

DOCTYPE Identifies the declaration as a document type declaration

DM Specifies that the source file contains tags used to define dialog elements for a Dialog Manager application

SYSTEM Indicates that the syntax rules for defining elements are contained in an external file

> Closes the markup declaration.

External files that are embedded (through the use of entity declarations) within the source file intended for compilation cannot contain a DOCTYPE declaration. They are compiled using the DOCTYPE declaration of the source file they are embedded within. For more information on entity declarations and embedding external files within source files, see "Entities and Parameter Entities" on page 2-6.

Including Comments in Your Markup

If you want to include notes, reminders, or other text that you don't want processed in your source files, you can insert them as comments, and the compiler ignores them.

Note: You cannot place comments within any of the DTL tags. A comment placed within a start or end tag will cause the tag to end, and the text following the comment will be treated as part of the tag content.

Like document type declarations, comments must be enclosed within markup declaration delimiters (<! >). However, you must also delimit comments within markup declarations by preceding and following a comment with two dashes (--), like this:

```
<!-- This is the text of the comment -->
```

Because the dashes act as comment delimiters, you can use them in any markup declaration. For example, you can include a comment within a DOCTYPE declaration:

```
<!doctype dm system -- DECLARE DOCUMENT TYPE -->
```

However, you should be careful about how you use dashes within comments because unexpected results can occur if you do not code the dashes correctly.

For example, the following comment will generate an error because the second set of dashes is interpreted as the end of a comment and the text "Provides help for ordering" is treated as an additional markup declaration:

```
<!-- Panel DMH022 -- Provides help for ordering -->
```

If you delete one of the dashes in the second set of dashes, or use another symbol, no error will occur.

```
<!-- Panel DMH022 - Provides help for ordering -->
```

This block comment will produce an error because of the odd number of dashes in the first and last lines of the block:

```
<!----------------------------------------->
<!--This source file contains all of the -->
<!-- help panels for the application     -->
<!----------------------------------------->
```

We could avoid this problem by using a different symbol between the comment dashes, like this:

```
<!--**************************************-->
<!--This source file contains all of the -->
<!-- help panels for the application     -->
<!--**************************************-->
```

You cannot nest comments within other comments. You can, however, code multiple comments within a markup declaration, like this:

```
<!-- Here a comment --
-- THERE A COMMENT --
-- Everywhere a comment, COMMENT-->
```

As you can see, each of the comments begin and end correctly with the comment delimiters.

Entities and Parameter Entities

You can define, or *declare* frequently used words, phrases, and longer character strings in your source file as *entities* or *parameter entities* that represent text in the source file. You declare them within the DOCTYPE statement of your source file. After you declare them, you reference the names of the entities in place of the word or phrase in the text. This saves you time when marking up your text, and allows you to globally change the defined words or phrases in one place in the source file.

You can use entities and parameter entities for the following purposes:

- To replace single characters in text that are considered special characters. This can include characters not available on a particular keyboard, or characters that have special meaning to the compiler, such as the tag start delimiter (<), that you want to treat as normal text.

 DTL provides you with a set of predefined single-character entities. See "Predefined Entities" on page 2-12 for a list of these entities.

- To replace strings of text, such as words, phrases, and longer text strings used frequently in the source file text.

- To embed entire files in a source file. This is useful for breaking up a source file into smaller, more manageable files, and for declaring entities that are shared by different source files.

When you reference an entity or parameter entity in the text of a source file, you must use certain symbols to delimit the reference.

- Precede entity references with an ampersand (&) and follow them with a semicolon (;) or a blank space.

- Precede parameter entity references with a percent sign (%) and follow them with a semicolon (;) or a blank space.

Entities

Entities are symbolic statements that represent text strings in a source file. Like other markup declarations, entity declarations must be enclosed within markup declaration delimiters (<! >). In addition, you must place entity declarations, within the *declaration subset* of the DOCTYPE statement. The declaration subset is delimited by left and right brackets ([]) and is coded within the DOCTYPE statement.

Within the markup declaration delimiters, you declare the entity with the term "entity," the name you are assigning to the entity, and the text string the name represents. The text string of the entity must be enclosed in double quotes.

```
<!doctype dm system [
<!entity name "text string">
]>
```

Entity names must have the following characteristics:

- One to eight characters in length
- The first character must be alphabetic (a−z or A−Z)
- Remaining characters, if any, can be from A−Z, a−z, 0−9, a hyphen (−), or a period (.)
- Entity names are case-sensitive.

This example declares an entity named "guar" for the phrase "full, unconditional, money-back guarantee":

```
<!doctype dm system [
<!entity guar "full, unconditional, money-back guarantee">
]>
```

Now that we've declared the entity, we can use the entity name in our source file text instead of the entire text string. To specify an entity name in text, you must precede the name with an ampersand (&) and follow it with a semicolon (;) or a blank, as we did in this panel text:

```
<!doctype dm system [
<!entity guar "full, unconditional, money-back guarantee">
]>

<panel name=widgets width=40 depth=15>Widgets
  <area>
    <info width=36>
      <p>You'll love the wide selection of merchandise
      in our Widgets department.
      <p>And, like all of our merchandise, Widgets come
      with our &guar;.
    </info>
  </area>
</panel>
```

As long as we declared the entity properly, the compiler recognizes the entity reference in the source file and replaces it with the text of the entity declaration. Figure 2-1 on page 2-8 shows the result.

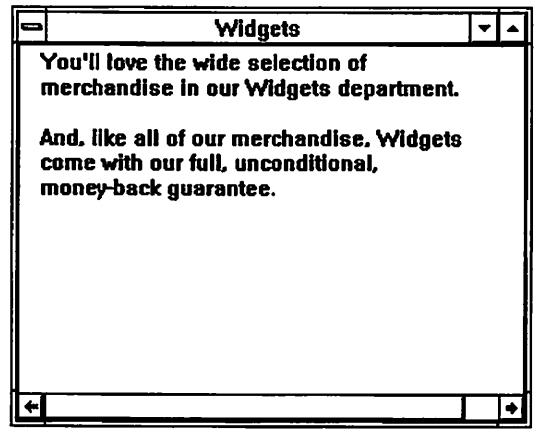


Figure 2-1. Entity Reference for Text Substitution

We can reference the same entity in the text of the source file as many times as we like. And, if we should ever want to change the text of the entity, we only have to do it in one place—where we declared it in the declaration subset.

We'll change the entity we declared earlier to show you what we mean.

```
<!doctype dm system [
<!entity guar "partial, conditional, non-refundable guarantee">
]>

<panel name=widgets width=40 depth=15>Widgets
  <area>
    <info width=36>
      <p>You'll love the wide selection of merchandise
      in our Widgets department.
      <p>And, like all of our merchandise, Widgets come
      with our &guar;.
    </info>
  </area>
</panel>
```

The only change we made was to the text of the entity declaration, not the entity name. Following reformatting, the text of the entity reference now looks like this:

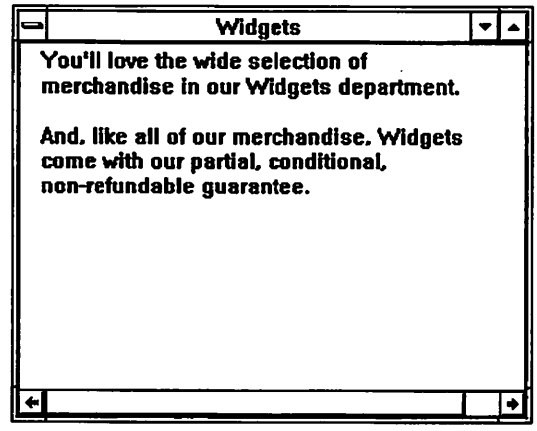


Figure 2-2. Entity Reference for Text Substitution

If, for any reason you need to change the name of an entity, be sure to update all of the references to the entity name in your text.

You can also define the text of an entity in an external file and reference that file in an entity declaration. If you do this, you must include the SYSTEM parameter in the entity declaration, to indicate to the compiler that the file is external. When the compiler encounters a reference to an external entity, it searches the current directory for the file. If the file is not found, the paths specified in the DPATH environment variable are searched.

For example, we'll define a text string we want to use as an entity in our source file in a file called WIDGETS.GML. Here are the contents of the WIDGETS.GML file:

```
doohickeys, whatnots, and gizmos
```

To declare this file in the entity declaration in our source file, we code it like this, with the SYSTEM parameter:

```
<!doctype dm system [
<!entity guar "full, unconditional, money-back guarantee">
<!entity widgets system>
]>
```

And, if we want to use the text string in our source file, we reference the entity "widgets" (in this case, the file name also serves as the entity name).

```
<!doctype dm system [
<!entity guar "full, unconditional, money-back guarantee">
<!entity widgets system>
]>

<panel name=widgets2 width=40 depth=15>More Widgets
  <area>
    <info width=36>
      <p>The fine selection of items in our Widgets department
      includes &widgets;.
      <p>And, like all of our merchandise, Widgets come with
      our &guar;.
    </info>
  </area>
</panel>
```

Figure 2-3 on page 2-10 shows the formatted result.

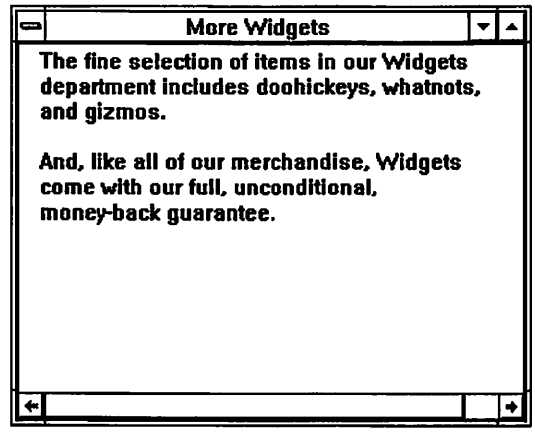


Figure 2-3. Entity Reference for Text Substitution and File Embedding

Anytime we want to update or change the text of the entity, we only need to change the text in the WIDGETS.GML file.

In the above example, the name "widgets" serves as the external file name and as the entity name. That's because a SYSTEM entity defaults to the entity name with .GML appended as the extension. You can override this default by providing your own filespec and enclosing the external filespec within double quotes following the SYSTEM keyword.

For instance, if you want to declare a different entity name for the WIDGETS.GML file, "things" for example, code it like this in the entity declaration:

```
<!doctype dm system [
<!entity guar "full, unconditional, money-back guarantee">
<!entity things system "widgets.gml">
]>
```

And reference the entity name like this:

```
<panel name=widgets2 width=40 depth=15>More Widgets
  <area>
    <info  width=36>
      <p>The fine selection of items in our Widgets department
      includes &things;.
      <p>And, like all of our merchandise, Widgets come with
      our &guar;.
    </info>
  </area>
</panel>
```

The formatted result of this markup is the same as that shown in Figure 2-3, assuming no changes were made to the text of the WIDGETS.GML file.

Parameter Entities

Parameter entities allow you to place multiple entity declarations within an external file and reference them within a source file. To embed the entities into the source file, you must declare the external file as a parameter entity and reference it by preceding the filename with a percent symbol (%) and following it with a semicolon (;).

For example, we've declared all of our entities within an external file called SYMBOLS.SPE. Here are the contents of the SYMBOLS.SPE file:

```
<!ENTITY sb "ShelfBrowse">
<!ENTITY cotime "ten days">
<!ENTITY xcotime "five days">
<!ENTITY nttime  "three days">
<!ENTITY nitem "red checkout card">
<!ENTITY lfine "ten cents">
<!ENTITY cophone "555-1234">
```

We can embed the above file into the declaration subset of the source file with a parameter entity declaration and reference. As long as we declare the parameter entity and reference it properly, we can use any of the declared entities in the external file in the text of the source file.

```
<!doctype dm system
 [<!entity % SYMBOLS system> %SYMBOLS;]>

<panel name=chkout width=40 depth=18>Library Checkout Periods
  <area>
    <info width=36>
      <p>&sb; allows you to check out an inventory
      item for a maximum of &cotime;.
      However, you can renew the item for an additional
      &xcotime; by calling in your card number to our
      checkout phone line (&cophone;) any time of day.
      <p>If an inventory item is a new shelf item
      (indicated by the &nitem;), you may only reserve it for
      a maximum of &nttime;.
      You may not renew a new shelf item.
      <p>There is a fine of &lfine; per day for all
      items returned late.
    </info>
  </area>
</panel>
```

Figure 2-4 shows the formatted result.

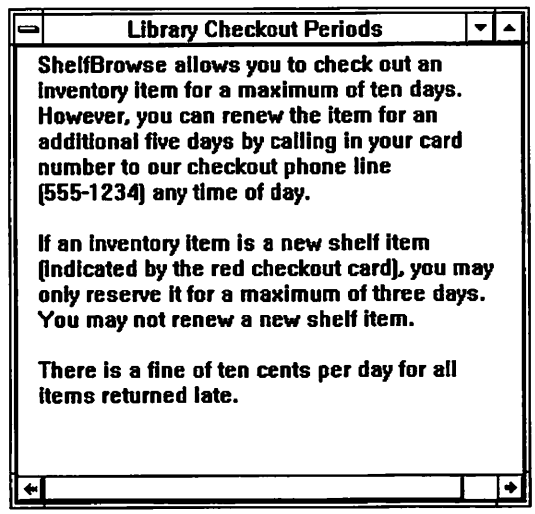


Figure 2-4. Parameter Entities

Parameter entity names must have the following characteristics:

- One to seven characters in length
- The first character must be alphabetic (a–z or A–Z)
- Remaining characters, if any, must be from A–Z, a–z, or 0–9
- Parameter entity names are case-sensitive.

Predefined Entities

The Dialog Tag Language provides you with a set of predefined entities that you can use in your source files. You can use them when the symbol you want is not present on your keyboard or conflicts with a compiler delimiter symbol.

You do not need to declare a predefined entity in order to use it. If you use the entity in your source file as you would an entity that you declare within your document subset, the compiler performs the translation for you.

The Dialog Tag Language predefined entities include:

&gtsym; greater than (>)
&ltsym; less than (<)
: colon (:)
& ampersand (&)
; semicolon (;)
. period (.)
&quote; single quote (')
&dquote; double quote (")
– short dash (–)
&us; underscore (_)
∨ logical or (|)
&sll; back slash (\)
&lbrk; left bracket ([)
&rbrk; right bracket (])
&lbrc; left brace ({)
&rbrc; right brace (})
− minus sign (–)
+ plus sign (+)

Chapter 3. Getting Started: Designing Application Panels

Each application panel you create will serve a specific purpose, with unique fields, messages, and help information defined for each one. In this chapter, we tell you how to define elements that are common among application panels. This includes defining the application panels, and the interactive elements of panels, including action bars, scrollable areas, and instruction text. We also tell you how to arrange the contents of your application panels using panel regions and dividers. The last section of this chapter tells you how to define command areas that are useful for applications that are migrated from a host environment.

Defining Application Panels: The PANEL Tag

You use the PANEL tag, its associated attributes, and the required PANEL end tag to define an application panel and the specific characteristics of the panel.

The PANEL start and end tags define the beginning and ending of an application panel. The PANEL start tag attributes define:

- Panel name
- Window title text
- Dimensions of the window that contains the panel
- Type of message the panel contains (if it is used to display a message)
- Associated key mapping list
- Name of the help panel for the application panel
- Initial cursor placement on the panel.

With the exception of the required NAME attribute used to identify the name of the application panel, all of the attributes for the PANEL tag are optional and contain default values that the Dialog Manager assumes if you do not specify the attribute. This section describes these attributes, and how to use them.

The PANEL start and end tags look like this, respectively:

```
<panel name=mainpan>
:
</panel>
```

In the preceding example, we included the required NAME attribute and its value *mainpan* on the PANEL start tag. The Dialog Manager requires that each panel definition contain this attribute and an associated value in order to identify the panel. The panel name is used as a parameter on the Dialog Manager DISPLAY service call to identify the panel to display. The panel name is also used as the panel ID when the panel ID is displayed.

The NAME value can be up to eight characters long, and the first character must be alphabetic. All additional characters for the NAME value must be either alphabetic (A–Z or a–z), or numeric (0–9).

The Window Title

The text that appears in the title bar of the window that contains a panel is called the title text. You define the title text by coding it as tag content for the PANEL start tag.

This example uses the text Catalog Ordering System as title text:

```
<panel name=mainpan>Catalog Ordering System
⋮
</panel>
```

If the panel ID is visible, it appears to the left of the window title and is separated from the window title with a hyphen (-).

Panel Design Note

As recommended by SAA Common User Access, the title text for a primary window should contain the application name. You can also change the title text of the primary window at run time by updating the ZWINTTL system variable. For a complete description of the ZWINTTL system variable, refer to Chapter 16. "System Variables" in the *Dialog Manager Guide and Reference*.

The title text for a pop-up window should be the name of the action that is performed in the pop-up. For example, if your application displays a pop-up window when the user chooses the **Open** action from the action bar, the title text for the pop-up window should be "Open." If you do not specify the title text on the PANEL tag for a pop-up window, the Dialog Manager will use the title text from the primary window.

Window Size (Width and Depth)

The values of the WIDTH and DEPTH attributes are used only when the panel is the first panel displayed in the window. If the panel is not the first panel displayed in the window, Dialog Manager sizes the panel to fit into the current window. Thus, if a window already exists, the WIDTH and DEPTH values of the panel are ignored. Additionally, the user can size panels displayed in a primary window, which changes the width and depth of the panel.

The PANEL tag has a default WIDTH attribute of 76 characters and a default DEPTH attribute of 18 character units. These are the values the Dialog Manager assumes if you do not specify alternate dimensions for WIDTH and DEPTH.

The following example defines the window size as 60 characters wide and 15 character units deep:

```
<panel name=mainpan width=60 depth=15>Catalog Ordering System
⋮
</panel>
```

If we want the width of the window to be 76 characters wide (the default width), we only need to specify a value for DEPTH, as in this markup:

```
<panel name=mainpan depth=15>Catalog Ordering System
⋮
</panel>
```

This results in a window with a default width of 76 characters and a specified depth of 15 characters units.

Panel Design Note

When designing application panels, the values for WIDTH and DEPTH specify the dimensions of the panel. The width and depth you specify do not include the window title, action bar, window borders, or function key area. These items (if they exist in the panel definition), are added to the width and depth you specify.

Message Type

If you intend for an application panel to display messages, you can use the optional MSGTYPE attribute to define the severity of the message that is displayed in the panel. By specifying the correct message severity, you ensure that the alarm will be sounded, if necessary, for the MSGTYPE you specify. The tone that the Dialog Manager uses for the alarm depends on the MSGTYPE value.

The values you can assign to the MSGTYPE attribute are:

INFO To define the message panel as an information message. Information messages tell the user that a function is performing normally, or has performed normally. INFO is the default MSGTYPE value.

WARNING To define the message panel as a warning message. Warning messages tell the user that a potentially undesirable situation could occur. The Dialog Manager sounds an audible alarm when a warning message appears.

ACTION To define the message panel as an action message. Action messages tell the user that an exception condition has occurred. The user must perform an action to correct the situation. The Dialog Manager sounds an audible alarm when an action message appears.

For a complete description of how to display messages, refer to Chapter 5, "Using Message Facilities" of the *Dialog Manager Guide and Reference*.

Key Mapping Lists

To specify the function keys that are active for an application panel, use the KEYLIST attribute of the PANEL tag. This attribute specifies the name of a key mapping list you define for the application. A key mapping list contains the keys that are active while the panel is displayed. The key mapping list also specifies what command is run when each key is pressed.

This PANEL definition references a key mapping list named *key01*:

```
<panel name=mainpan keylist=key01>Catalog Ordering System
:
</panel>
```

For more information on defining key mapping lists, see Chapter 10, "Defining Key Mapping Lists" on page 10-1.

Associated Help Panels

To provide additional help for an application panel, identify the name of the associated help panel with the HELP attribute of the PANEL tag. The help panel you specify appears when the user selects extended help while in the application panel or when contextual help is selected for an item on the panel, but no contextual help is available for the item.

This panel definition references a help panel named *ordhelp*:

```
<panel name=mainpan help=ordhelp>Main Ordering Panel
:
</panel>
```

Chapter 7, "Help Panels" on page 7-1 tells you how to create help panels for your application.

Cursor Placement

The PANEL tag attributes, CURSOR, CSRINDEX, and CSRPOS, allow you to specify where the cursor is placed when the panel is initially displayed. If you do not specify a specific cursor position, Dialog Manager places the cursor in the first field in the PANEL definition that can contain the cursor.

Use the CURSOR attribute to specify the field that contains the initial cursor setting. Use the CSRINDEX and CSRPOS attributes to identify positions within the field you specify with the CURSOR attribute. CSRINDEX and CSRPOS can only be used when the CURSOR attribute is used.

CURSOR attribute: To identify a particular field that is to contain the initial cursor setting, use the CURSOR attribute to specify the value of the NAME attribute from one of the following tags:

CHOICE The cursor appears in the specified selection field choice.

DTAFLD The cursor appears in the first byte position of the data field. Cursor positioning is valid only within INPUT or BOTH type data fields.

LSTCOL The cursor appears in the first row in the list column. Cursor positioning is valid only within INPUT or BOTH type list columns.

LSTFLD The cursor appears in the first INPUT or BOTH type column in the list field.

SELFLD The cursor appears on the first selected choice in the selection field, or on the first choice, if none of the choices are selected.

SELLST The cursor appears on the first selected item in the selection list.

UC The cursor appears in the specified user control.

Chapter 5, "Application Panel Fields" on page 5-1 provides a complete description of the types of interactive fields you can define for your application panels.

The CURSOR Attribute

In the following example, the CURSOR attribute specifies the data field NAME *placedfd*. When the panel is initially displayed, the cursor appears in the first

character position of that field. Figure 3-1 on page 3-5 shows the formatted result.[1]

```
<panel name=mainpan width=45 depth=15 cursor=placedfd>Travel Agency
  <ab>
⋮
  </ab>
  <area>
    <selfld>Destinations:
      <choice>London
      <choice>Madrid
      <choice>Paris
      <choice>Zurich
    </selfld>
    <divider>
    <dtafld name=placedfd datavar=place>Other
  </area>
</panel>
```

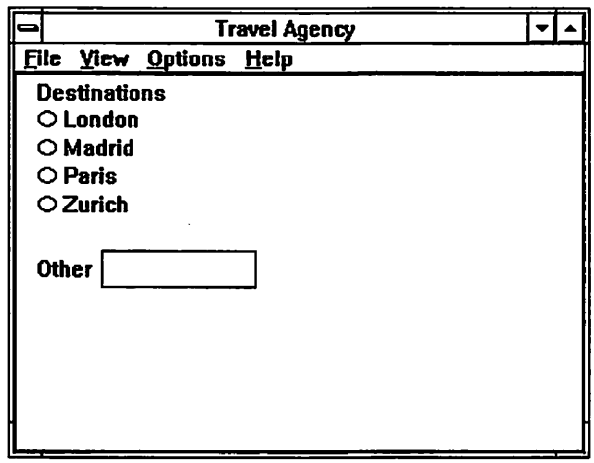


Figure 3-1. Cursor Placement

If no cursor placement was specified for the preceding example, the cursor would appear on the **London** choice of the **Destinations** selection field when the panel is initially displayed.

If you want the initial placement of the cursor to appear within a command area you have defined for a panel, use the value CMDAREA with the CURSOR attribute, as in this example:

```
<panel name=mainpan cursor=cmdarea>
⋮
  <cmdarea>
</panel>
```

"Defining a Command Area" on page 3-25 provides a complete description of the CMDAREA tag.

[1] In this example, and in other examples in this chapter, we show tag markup for elements such as fields and variables that have not been discussed yet so that we can illustrate the formatting characteristics of some tags. The syntax of these elements are not important for the purposes of these examples. We discuss the syntax conventions of these elements in later chapters of this book.

The CSRINDEX Attribute

To place the cursor in a particular row within a list field, use the CURSOR attribute to specify the name of the list field or a particular column within the list field, and the CSRINDEX attribute to specify the row within the list field or list column where the cursor should be placed.

The value you assign to CSRINDEX must be numeric. The numeric value indicates which row in the list column should contain the cursor when the panel is initially displayed.

The CSRPOS Attribute

If you use the CURSOR attribute to place the cursor within an input-only, or input/output data field or list column, or the command area, you can also define a specific byte position for the cursor using the CSRPOS attribute.

The value you assign to the CSRPOS attribute must be numeric. This numeric value indicates the number of byte positions from the left margin of the field where the cursor is placed, where a 1 specifies that the cursor should be in front of the first byte position.

Panel Design Note

You can also specify cursor placement at run time, using parameters available on the Dialog Manager DISPLAY service call. If used, the cursor placement parameters on the DISPLAY call override the cursor placement attributes specified on the PANEL tag.

Chapter 14, "Dialog Manager Services" in the *Dialog Manager Guide and Reference* provides a complete description of the DISPLAY service call.

Defining Action Bars and Pull-Downs

In order to create a consistent user interface, you should design your applications according to the object-action process sequence defined by the SAA Common User Access. The action bar is a major user interface component that helps you achieve consistency in your applications.

The action bar is the panel element located at the top of an application panel that contains action bar choices for the panel. Each action bar choice represents a group of related actions that appear in the pull-down menu associated with the action bar choice. When the user selects an action bar choice, the associated pull-down menu appears directly below the action bar choice. Pull-down menus contain choices that, when selected by the user, perform immediate actions that apply to the contents of the panel.

Panel Design Note

Dialog Manager and DTL provide the tools to help you create the object-action process sequence in your application, but it is your responsibility as an application designer to ensure that the contents of your action bar are actions that can be applied to the objects contained within your panel.

Typically, application panels intended for display within primary windows contain action bars that present the user with all of the available actions that apply to that panel. Application panels that are displayed within pop-up windows should use pushbuttons (immediate-action selection fields) rather than action bars to present the user with the actions that apply to that panel.

Figure 3-2 shows a Common User Access-conforming action bar, with the standard **File** pull-down menu displayed.

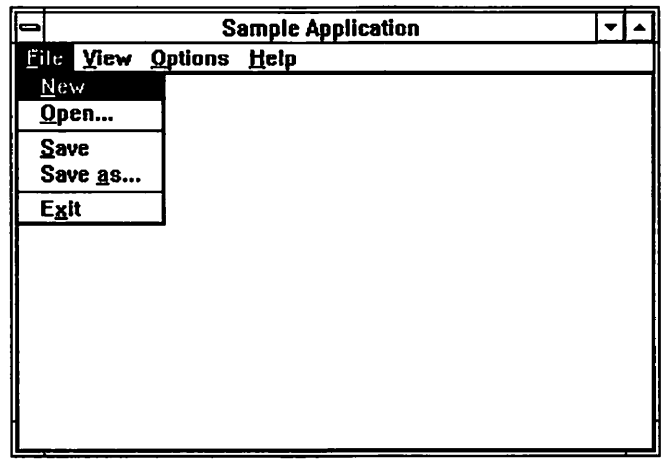


Figure 3-2. Action Bar and Pull-Down

The tags you use to create the action bar and its pull-down menus are:

AB To start an action bar definition. The required AB end tag ends an action bar definition.

ABC To define each of the action bar choices.

ACTION To specify an action to be taken when the pull-down choice is selected. The ACTION tag is coded within the PDC tag.

M To specify the mnemonic character for an action bar choice or pull-down choice.

PDC To define the choices on the pull-downs associated with each action bar choice.

PDSEP To create a horizontal divider line in a pull-down.

The following list describes how to code an action bar definition:

- Code the AB start tag immediately after the PANEL start tag and before any other tags in the panel.
- Following the AB start tag, code an ABC tag for each action bar choice in the action bar. The text you specify on the ABC tag is the text that appears in the action bar as the action bar choice.

- Code the associated PDC tags within the ABC tags. The text you specify on the PDC tag is the text that appears as the pull-down choice.
- If you want a horizontal divider line in the pull-down, code the PDSEP tag between the pull-down choices (PDC tags) that you want the line to appear between.
- Following each PDC tag, code one or more ACTION tags to specify what type of action occurs when that pull-down choice is selected by the user.

 The ACTION tag RUN attribute (and a matching value) define an immediate action for the pull-down choice. If you define multiple ACTION tags for a pull-down choice, one of which contains a RUN value, code the RUN action last, because any actions specified after a RUN action are ignored.
- Use the M tag to define mnemonics for the action bar choices and pull-down choices.
- End the action bar definition with the required AB end tag.

In addition to the steps outlined in the preceding list, you should define the variable declarations and variable classes associated with the action bar at the beginning of the source file. Chapter 4, "Variables and Variable Classes" on page 4-1 tells you how to define variable declarations and classes for your application.

The following example shows the markup for the action bar shown in Figure 3-2 on page 3-7. The detailed markup for the **File** pull-down is included.

```
<panel name=mainpan width=50 depth=15>Sample Application
  <ab>
    <abc><m>File
      <pdc><m>New
        <action run=New>
      <pdc><m>Open...
        <action run=Open>
      <pdsep>
      <pdc><m>Save
        <action run=Save>
      <pdc>Save <m>as...
        <action run=Saveas>
      <pdsep>
      <pdc>E<m>xit
        <action run=Exit>
    <abc><m>View

    .
    .
    .
    <abc><m>Options

    .
    .
    .
    <abc><m>Help

    .
    .
    .
  </ab>
  .
  .
  .
</panel>
```

Pull-Down Choice Actions

A pull-down choice provides an immediate action to the user. To ensure that a pull-down choice performs an immediate action, you must code an ACTION tag that specifies the RUN attribute for each pull-down choice. The value you assign to RUN tells the Dialog Manager which command to run when the user selects the choice.

In the preceding markup example, each ACTION definition uses the RUN attribute to specify a command. Each of these commands must be defined within a command table for the application. Chapter 9, "The Application Command Table" on page 9-1 tells you how to define commands for an application.

In addition to the RUN action, you can specify other types of actions to occur when a pull-down choice is selected. The SETVAR and TOGVAR attributes on the ACTION tag can be used to set and toggle variables. This is useful when the variable being changed is shared with another field on the panel.

For example, you can de-emphasize a selection field choice based on an action that was selected from the action bar. By using the SETVAR attribute of ACTION to set a variable that is also specified with the AVAILVAR attribute of a selection field CHOICE tag, you can dynamically de-emphasize the selection field choice when the pull-down choice is selected. See "Defining Selection Fields" on page 5-13 for more information on selection fields.

Remember, any SETVAR or TOGVAR actions for a pull-down choice must be coded before any ACTION definitions specifying the RUN action, since actions coded after RUN are ignored.

Mnemonics

Each action bar and pull-down choice has an associated mnemonic character defined with the M tag. When you add an M tag to the text of an action bar or pull-down choice, it designates the character that immediately follows it as the mnemonic for that choice, and underscores it on the user's display. The markup example on page 3-8 shows how the mnemonic characters for the action bar in Figure 3-2 on page 3-7 are defined.

When you specify mnemonics for the choices in an action bar, the user can select that choice from anywhere within the action bar by typing the mnemonic.

In order to provide a consistent user interface, you must specify a mnemonic for every action bar choice or pull-down choice within the action bar definition or specify no mnemonics within the action bar at all. For this reason, if you specify mnemonics, each mnemonic character within an action bar definition must be unique.

In this markup example, we've highlighted the characters designated as mnemonics, in addition to the mnemonic tags, to show you the specified mnemonic characters.

```
<panel name=mainpan width=50 depth=15>Sample Application
  <ab>
    <abc><m>File

    <abc><m>View

    <abc><m>Options

    <abc><m>Help

  </ab>
</panel>
```

Figure 3-2 on page 3-7 shows the formatted mnemonic characters.

Action Bar Help

You can provide help for each action bar choice and pull-down choice with the HELP attribute on the ABC and PDC tags, respectively. By specifying the name of a help panel for the action bar choice or pull-down choice, the Dialog Manager knows which help panel to display when the user selects help on that choice. If you do not specify help for a pull-down choice, the help for the action bar choice is displayed, when the user selects help. If there is no help defined for the action bar choice, the extended help panel is displayed.

In the following example, we've added the HELP attribute (and a value) to each of the action bar choices and pull-down choices in the action bar defined on page 3-8. The values specified with each HELP attribute are the NAME values of defined help panels.

```
<panel name=mainpan width=50 depth=15>Sample Application
  <ab>
    <abc help=hfile><m>File
      <pdc help=hnew><m>New
        <action run=New>
      <pdc help=hopen><m>Open...
        <action run=Open>
      <pdsep>
      <pdc help=hsave><m>Save
        <action run=Save>
      <pdc help=hsaveas>Save <m>as...
        <action run=Saveas>
      <pdsep>
      <pdc help=hexit>E<m>xit
        <action run=Exit>
    <abc help=hview><m>View

⋮
    <abc help=hoption><m>Options

⋮
    <abc help=ispnabc><m>Help

⋮
  </ab>
⋮
</panel>
```

The HELP value assigned to the **Help** choice in the preceding example (*ispnabc*) is the help panel provided by Dialog Manager. The following section describes the Help action bar pull-down.

Help Action Bar Pull-down

Common User Access requires that you put the **Help** action bar choice as the last action bar choice in the action bar definition. Common User Access also recommends the pull-down choices that should be in the Help pull-down.

To assist you in defining your Help pull-down, DTL provides you with a sample markup file for the Help pull-down that is consistent with the format recommended by the Common User Access. This file is called ABCHELP.GML. You can make a copy of this file and tailor it for use in your application.

Specifically, you may want to:

- Make the use of mnemonics consistent with all of the other choices on your application action bar. "Mnemonics" on page 3-9 tells you how to do this.
- Add function to your application for the two Common User Access-defined pull-down choices not supported directly by the Dialog Manager, **Tutorial** and **About**. In addition to supporting the commands associated with the two choices, you should also develop help panels for each choice.

The following example shows the contents of the ABCHELP.GML file. Included in the file are specifications for help panels for each of the Dialog Manager supported commands.

```
<abc help=ispnabc><m>Help
  <pdc help=ispnhph><m>Help for help...
    <action run=helphelp>
  <pdc help=ispnexhe><m>Extended help...
    <action run=exhelp>
  <pdc help=ispnkeys><m>Keys help...
    <action run=keys>
  <pdc help=ispnindx>Help <m>index...
    <action run=index>
  <pdc><m>Tutorial...
    <action run=tutorial>
  <pdsep>
  <pdc><m>About...
    <action run=logo>
</abc>
```

Unavailable Pull-Down Choices

Unavailable emphasis is a visual cue to the user that a particular choice cannot be selected because of some condition in the application. The Dialog Manager allows you to show unavailable emphasis for a pull-down choice by using the AVAILVAR attribute on the PDC tag.

The value you assign to the AVAILVAR attribute is the name of a variable that the Dialog Manager evaluates at run time to determine the availability of a pull-down choice. If the variable is set to true (1), the pull-down choice is displayed as an available choice and can be selected by the user. If the variable is set to false (0), then the pull-down choice is displayed with unavailable emphasis (grayed), and cannot be selected by the user.

If you do not use the AVAILVAR attribute, the pull-down choice is displayed as available.

To illustrate the concept of unavailable emphasis, assume that the action bar in the following markup example belongs to a library system application that is used to maintain control over a set of files. While viewing a particular file, the user may want to check the file out of the library. If the file is already checked out, the pull-down choice to check out the file should be unavailable, as a visual cue to the user that the file cannot be checked out at that time.

To de-emphasize the **Checkout** pull-down choice, we used the AVAILVAR attribute to specify the name of a variable that is set to false when the file is checked out. When the file is checked back into the library, the AVAILVAR variable is set to true, so the **Checkout** choice is once again displayed as available.

```
<!doctype dm system>

<varclass name=covar type='char 1'>

<varlist>
  <vardcl name=out varclass=covar>
</varlist>

<panel name=mainpan width=50 depth=15>Sample Application
  <ab>
    <abc><m>File

⋮
    <abc><m>View

⋮
    <abc><m>Options
      <pdc><m>Copy
        <action run=copy>
      <pdc availvar=out>Chec<m>kout
        <action run=out>
      <pdc>Checki<m>n
        <action run=in>
    <abc><m>Help

⋮
 </ab>
⋮
</panel>
```

We added the variable list and variable class associated with the AVAILVAR value to illustrate how the variable is declared. For more information on variables and variable classes, see Chapter 4, "Variables and Variable Classes" on page 4-1.

Figure 3-3 shows how the unavailable choice appears in the pull-down.

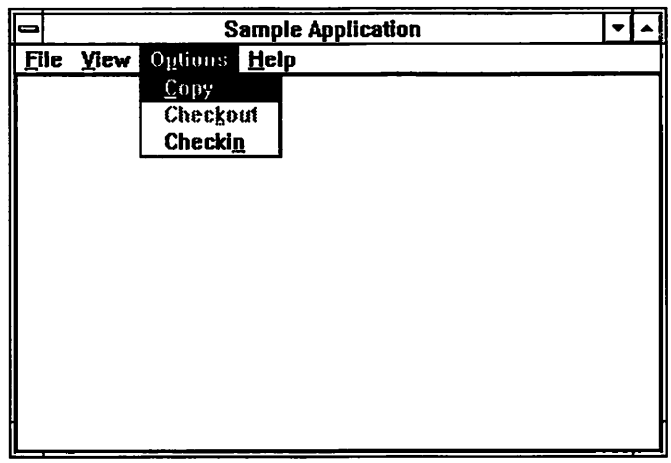


Figure 3-3. Pull-Down Choice (de-emphasized)

Check Marks for Pull-Down Choices

Current state indicators are used to indicate which pull-down choices are currently selected, or *active*. Current state indicators in pull-down choices are check marks to the left of the pull-down choice text.

You specify check marks for pull-down choices with the CHECKVAR and MATCH attributes of the PDC tag. The CHECKVAR attribute specifies the name of a variable that you set at run time to indicate if the pull-down choice should be checked. If the CHECKVAR variable is equal to the MATCH value, the pull-down choice is checked.

Continuing with the library application, assume that the user can view the files in the library sorted by name, owner, date, or size. The check mark indicating the current sort option being used provides a visual cue to the user.

To place a check mark next to any of the pull-down choices, the same CHECKVAR value is specified for each choice, and a unique MATCH value is specified for each choice. The application sets the variable specified with CHECKVAR to the MATCH value to indicate the sorting option being used. The variable specified with CHECKVAR is changed each time the sorting option is changed. This provides a visual reminder to the user of how the files are sorted.

```
<!doctype dm system>

<varclass name=sortvar type='char 1'>

<varlist>
  <vardcl name=sorttype varclass=sortvar>
</varlist>

<panel name=mainpan width=50 depth=15>Sample Application
  <ab>
    <abc><m>File

⋮
    <abc><m>View
      <pdc checkvar=sorttype match=N><m>Name
        <action run=name>
      <pdc checkvar=sorttype match=O><m>Owner
        <action run=owner>
      <pdc checkvar=sorttype match=D><m>Date
        <action run=date>
      <pdc checkvar=sorttype match=S><m>Size
        <action run=size>
    <abc><m>Options

⋮
    <abc><m>Help

⋮
  </ab>
⋮
</panel>
```

Figure 3-4 on page 3-15 shows how the **View** pull-down would appear in this scenario.

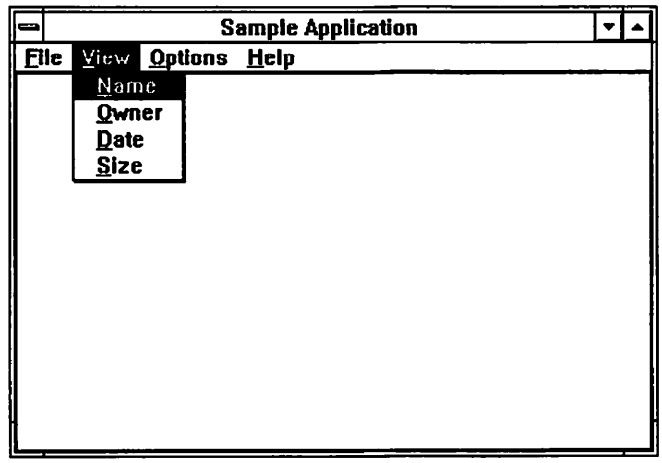


Figure 3-4. Pull-Down Choice with Check Mark

Defining a Scrollable Area: The AREA Tag

Many of the panels you create are displayed in primary windows that can be sized by the user. Since the user can control the size of the window, you should define the panel as *scrollable* so the user can view all of the information on the panel. You define a panel as scrollable by coding the AREA tag and its required end tag within a PANEL definition.

If you define a scrollable area on a panel and the contents of the panel body do not fit within the current window dimensions, Dialog Manager places scroll bars in the scrollable area of the panel. The scroll bars indicate to the user that more information is available and the user can view the contents of the panel by scrolling through it.

When a panel contains a scrollable area, the vertical, horizontal, or both scroll bars become visible on the panel, as needed, to allow the user to scroll the contents that are not visible into view. The scroll bars are not visible if all of the panel contents can be viewed—an indication to the user that there is no more information to be seen.

You code an AREA definition within a PANEL definition like this:

```
<panel name=mainpan>Application Name
  <ab>

:
  </ab>
  <area>

:
  </area>
</panel>
```

All of the markup contained within an AREA definition can be accessed by the user through scrolling. If you do not define a scrollable area on the panel, and the panel contents do not fit within the current window dimensions, the user has to size the window to view the information not currently visible.

Figure 3-5 on page 3-16 shows how the scroll bars appear when the contents of a panel do not fit inside the current window dimensions. Figure 3-6 on page 3-16 shows the same panel sized so that all of the panel is visible.

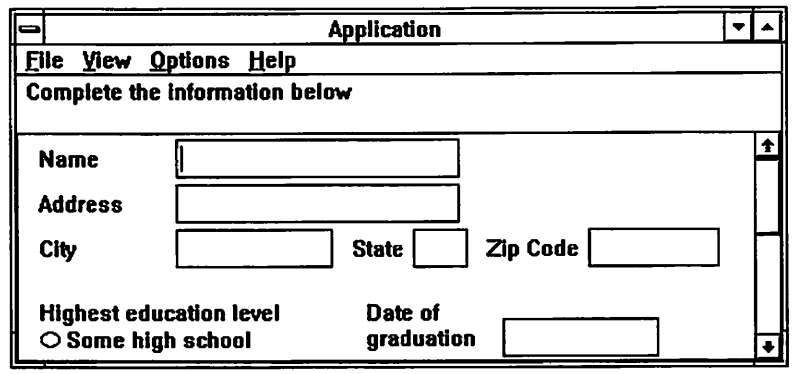


Figure 3-5. Application Window with Scroll Bars

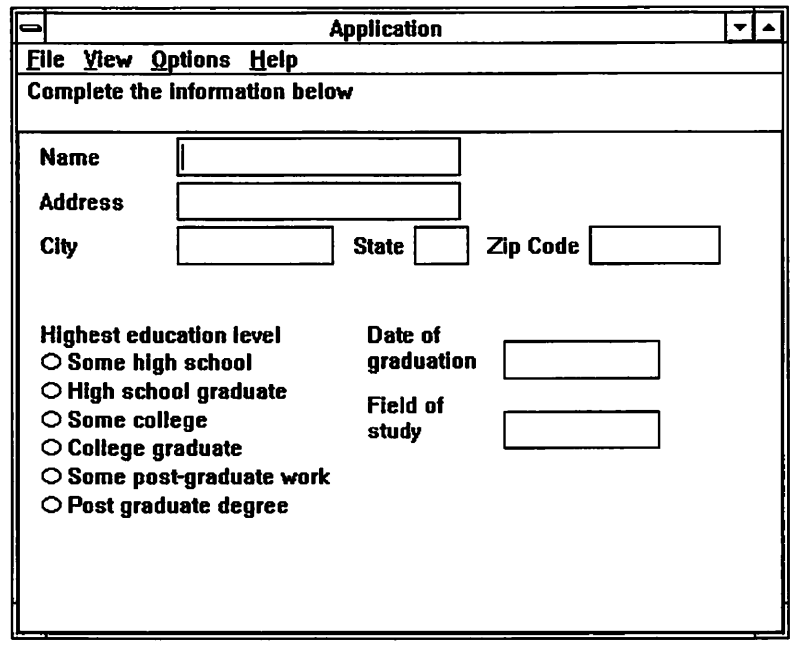


Figure 3-6. Application Window Sized Larger

You can define the size of the outside margins of a scrollable area with the MARGINW and MARGIND attributes of the AREA tag. The MARGINW attribute has a default value of 1, which specifies a one-character margin for the AREA width. The MARGIND attribute has a default value of 0, which specifies that no margin is created for the AREA depth.

Use the MARGINW attribute to increase the margins on the sides of a scrollable area. For example, the panel in Figure 3-7 on page 3-17 contains a small selection field. Notice how the selection field is formatted along the left margin of the scrollable area.

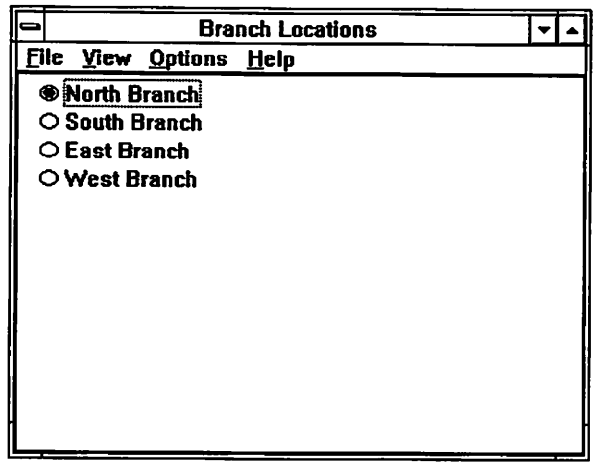


Figure 3-7. Selection Field in a Scrollable Area

Here is the markup for the panel in Figure 3-7:

```
<panel name=smallfld width=45 depth=15>Branch Locations
  <ab>

:
  </ab>
  <area>
    <selfld>
      <choice>North Branch
      <choice>South Branch
      <choice>East Branch
      <choice>West Branch
    </selfld>
  </area>
</panel>
```

We can use the MARGINW attribute to increase the margin widths on the left and right sides of the scrollable area:

```
<panel name=smallfld width=45 depth=15>Branch Locations
  <ab>

:
  </ab>
  <area marginw=10>
    <selfld>
      <choice>North Branch
      <choice>South Branch
      <choice>East Branch
      <choice>West Branch
    </selfld>
  </area>
</panel>
```

We specified a MARGINW value of 10 for the AREA definition. Figure 3-8 on page 3-18 shows the result.

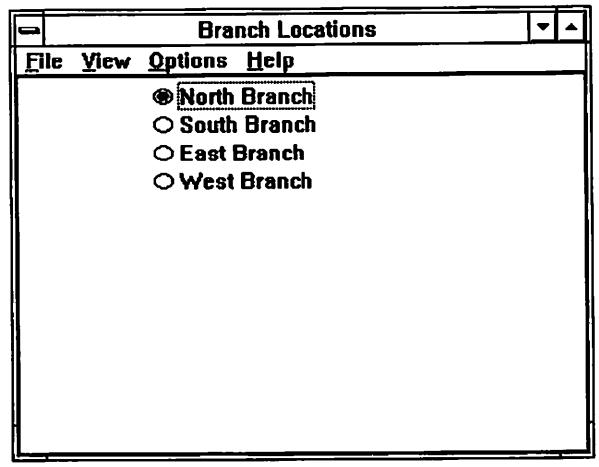


Figure 3-8. Scrollable Area (MARGINW=10)

To increase the margins on the top and bottom of a scrollable area, use the MARGIND attribute to specify the additional depth in character units. We'll add a MARGIND value of 3 to the markup for the panel in Figure 3-8 to show you how it works.

```
<panel name=smallfld width=45 depth=15>Branch Locations
  <ab>

⋮
  </ab>
  <area marginw=10 margind=3>
    <selfld>
      <choice>North Branch
      <choice>South Branch
      <choice>East Branch
      <choice>West Branch
    </selfld>
  </area>
</panel>
```

Figure 3-9 shows the result.

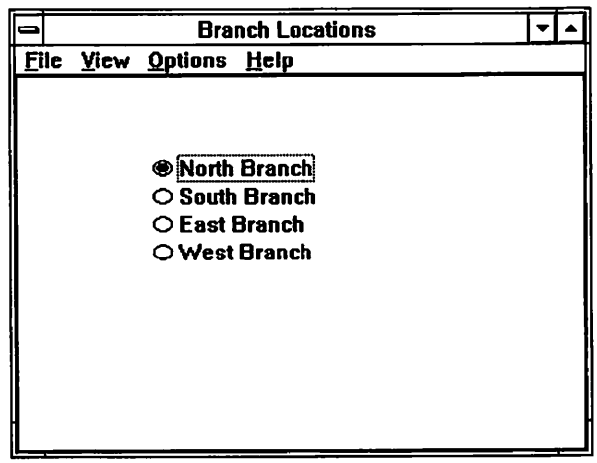


Figure 3-9. Scrollable Area (MARGIND=3)

Because of the nature of panels and how you define them for use within windowed environments, the AREA tag can be a useful tool for your Dialog Manager applications. Throughout this book, you'll find many more examples of how to use the AREA tag.

Defining Top and Bottom Instructions

DTL provides you with two tags to define top and bottom instructions for your application panels, the TOPINST and BOTINST tags. Neither of the tags have required end tags or attributes associated with them.

Use the instruction tags to define text that tells the user how to interact with the panel or how to continue with an application. Since you must code the instruction tags outside of the scrollable area of the panel, they always remain visible in the window. Code the TOPINST tag immediately after the action bar definition (or the PANEL start tag if the panel does not contain an action bar). Code the BOTINST following the main body of the panel, before the PANEL end tag.

This application panel markup contains both types of instructions. Figure 3-10 shows the results.

```
<panel name=mainpan width=45 depth=15>Item Selection
  <ab>

:
  </ab>
  <topinst>Select one of the following items and Enter.
  <area>
    <selfld>
      <choice>Automotive
      <choice>Hardware
      <choice>Health and beauty
      <choice>Lawn and garden
      <choice>Sporting goods
    </selfld>
  </area>
  <botinst>To exit the application, press F3.
</panel>
```

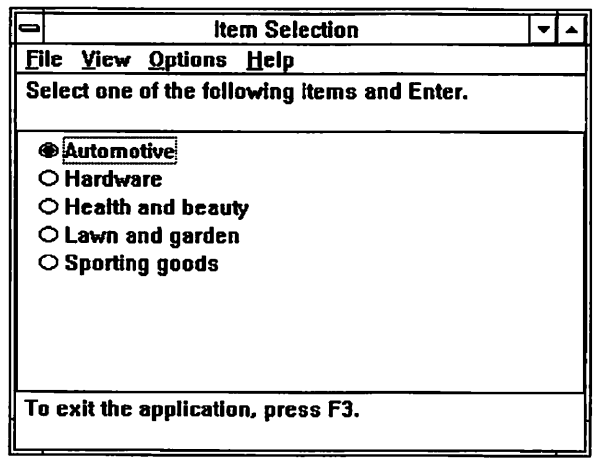


Figure 3-10. Top and Bottom Instructions

Defining Panel Layout: The DIVIDER and REGION Tags

DTL provides you with a way to specify the arrangement of the elements on your panel, and a way to add spaces and divider lines, so that your panels are visually appealing. Use the REGION and DIVIDER tags to make your panels more usable and aesthetically pleasing.

The DIVIDER Tag

You can separate the elements on a panel or the regions you define for a panel with the DIVIDER tag. A DIVIDER definition produces either a blank or solid divider line, depending on the value you assign to the TYPE attribute of the DIVIDER tag. The default value, NONE, produces a blank divider line. The value SOLID produces a solid divider line.

There are two DIVIDER tags defined in the following example. The first DIVIDER does not specify a TYPE attribute, and produces a blank horizontal line. The second DIVIDER specifies a SOLID value, and produces a visible divider.

```
<panel name=fields width=45 depth=20>Selections
  <ab>

:
  </ab>
  <area>
    <selfld>Pick an item:
      <choice>Widget
      <choice>Doohickey
      <choice>Gizmo
    </selfld>
    <divider>
    <selfld>Pick a color:
      <choice>Red
      <choice>Green
    </selfld>
    <divider type=solid>
    <selfld>Pick a size:
      <choice>Miniscule
      <choice>Behemoth
    </selfld>
  </area>
  <botinst>To exit the application, press F3.
</panel>
```

Here's the result:

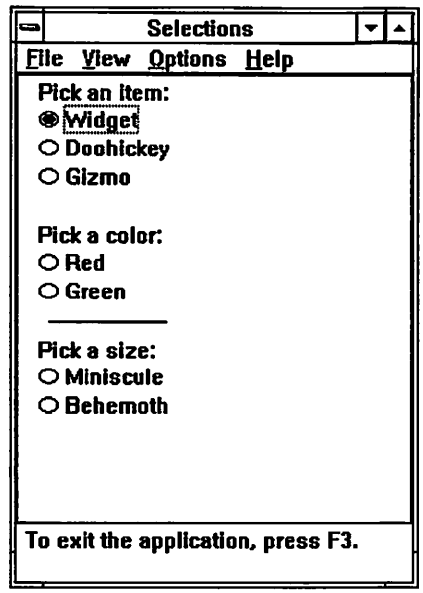


Figure 3-11. Area Dividers

The DIVIDER tag has two other attributes, GAP and GUTTER. GAP specifies whether the divider line completely crosses the panel area or region that contains the divider, or if a one-character gap is to remain at either end of the line. The valid values for the GAP attribute are YES (the default), and NO.

The value you assign to GUTTER specifies the size (in character units) of any additional space you want to reserve on either side of the line. The default is 1 character unit, because, even though the divider line itself is only one pixel wide, the Dialog Manager allots one character unit of screen space to the line. Any value above the default is split between the sides of the line. The GUTTER attribute is useful for creating blank space on a panel.

The REGION Tag

You can further define the areas of your panel, and how you want the information in the areas arranged, with the REGION tag. Using one or more regions within a PANEL definition provides an easy way of arranging the elements on a panel. Like PANEL and AREA tags, the REGION end tag is required.

The DIR (direction) attribute of the REGION tag specifies how the elements within a region are arranged, either horizontally or vertically. The default value is VERT, which arranges the elements within the region vertically. This means that if you do not specify a horizontal region (DIR=HORIZ), or if you do not define a region at all, the panel elements are arranged vertically by default.

For example, the selection fields in this region are arranged vertically, because no DIR value is defined for the region.

```
<panel name=fields width=45 depth=20>Selections
  <ab>

:
  </ab>
  <area>
    <region>
      <selfld>Pick an item:
        <choice>Widget
        <choice>Doohickey
        <choice>Gizmo
      </selfld>
      <divider>
      <selfld>Pick a color:
        <choice>Red
        <choice>Green
      </selfld>
      <divider>
      <selfld>Pick a size:
        <choice>Miniscule
        <choice>Behemoth
      </selfld>
    </region>
  </area>
  <botinst>To exit the application, press F3.
</panel>
```

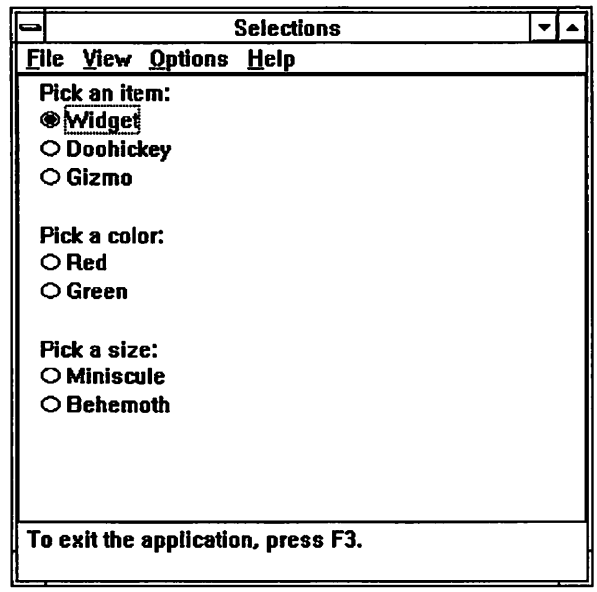


Figure 3-12. Vertical Region

We'll specify the HORIZ value for the region to change the layout of the selection fields to horizontal. Figure 3-13 on page 3-23 shows the result.

```
<panel name=fields width=45 depth=20>Selections
  <ab>

:
  </ab>
  <area>
    <region dir=horiz>
      <selfld>Pick an item:
        <choice>Widget
        <choice>Doohickey
        <choice>Gizmo
      </selfld>
      <divider>
      <selfld>Pick a color:
        <choice>Red
        <choice>Green
      </selfld>
      <divider>
      <selfld>Pick a size:
        <choice>Miniscule
        <choice>Behemoth
      </selfld>
    </region>
  </area>
  <botinst>To exit the application, press F3.
</panel>
```

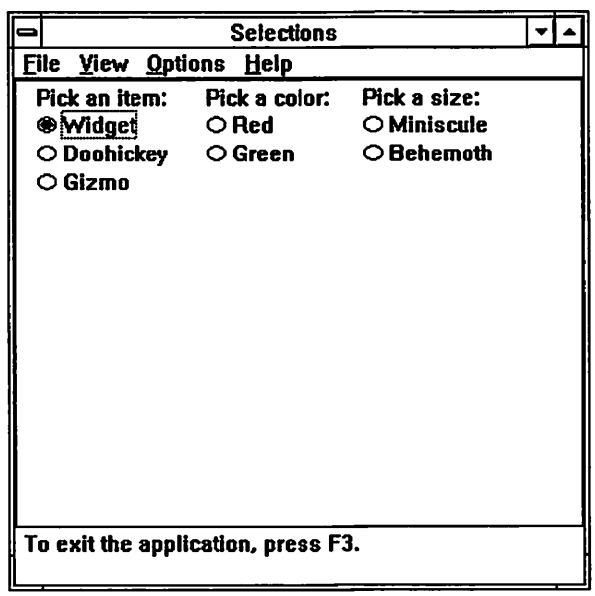


Figure 3-13. Horizontal Region

You can also code DIVIDER tags within REGION definitions. DTL defines the following formatting rules for DIVIDER tags within regions:

- A DIVIDER tag coded within a vertical region formats horizontally.
- A DIVIDER tag coded within a horizontal region formats vertically.

The following markup illustrates how REGION and DIVIDER tags format under different circumstances. This example shows both horizontal and vertical regions, as well as solid and blank dividers.

```
<panel name=mainpan width=50 depth=22>Application
  <ab>

:
  </ab>
  <topinst>Complete the information below and Enter.
  <area>
    <dtafld>Name
    <dtafld>Address
    <region dir=horiz>
      <dtafld>City
      <dtafld>State
      <dtafld>Zip code
    </region>
    <divider>
      <selfld>Age
         <choice> 0 - 12
         <choice>13 - 19
         <choice>20 - 29
         <choice>30 - 49
         <choice>50 - 64
         <choice>over 65
       </selfld>
    <divider type=solid>
    <region dir=horiz>
      <selfld>Highest education level
        <choice>Some high school
        <choice>High school graduate
        <choice>Some college
        <choice>College graduate
        <choice>Some post-graduate work
        <choice>Post graduate degree
      </selfld>
      <divider>
      <region>
        <info>
          <p>Complete if applicable:
        </info>
        <dtafld>Date of graduation
        <dtafld>Field of study
      </region>
    </region>
  </area>
</panel>
```

Figure 3-14 on page 3-25 shows how the preceding markup formats.

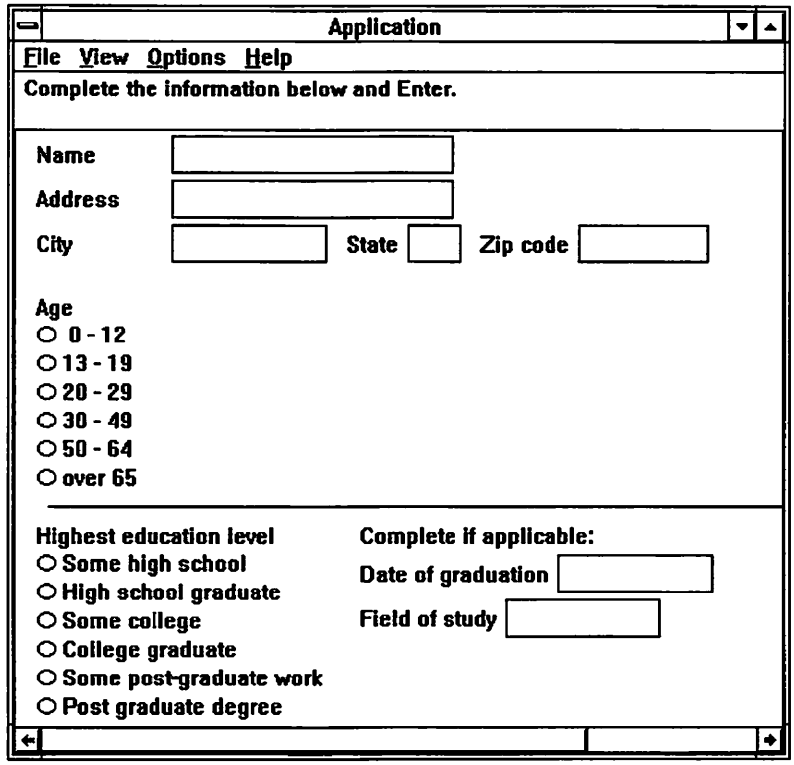


Figure 3-14. Horizontal Region

Notice that a vertical region is not needed for the entire panel area or for the selection fields. Both the scrollable area and selection fields format vertically by default, so there is no need to define a REGION for them.

In addition, the example contains nested regions. The data fields for entering the graduation date and field of study are arranged in a vertical region that is nested within a horizontal region.

Defining a Command Area

Many host applications are dependent on a command area in their panels. DTL provides you with a tag to define a command area for panels in applications that are going to be ported to host environments.

The command area contains an entry field and command prompt text, and is located at the bottom of an application panel. Users can enter commands in the command entry field. All commands entered into the command entry field are validated against the commands you define within the application command table and the DM-provided commands. For more information on the application command table, see Chapter 9, "The Application Command Table" on page 9-1.

You define a command area and specify the prompt-text with the CMDAREA tag. The Dialog Manager supplies the prompt symbol (===>) and reserves the entry field in the command area for user input.

You must code the CMDAREA tag immediately before the PANEL end tag.

```
<panel name=panel01 depth=15 width=45>Application Name
 <ab>

⋮
 </ab>
  <area>

⋮
  </area>
  <cmdarea>
</panel>
```

Here's how the command area formats on the panel:

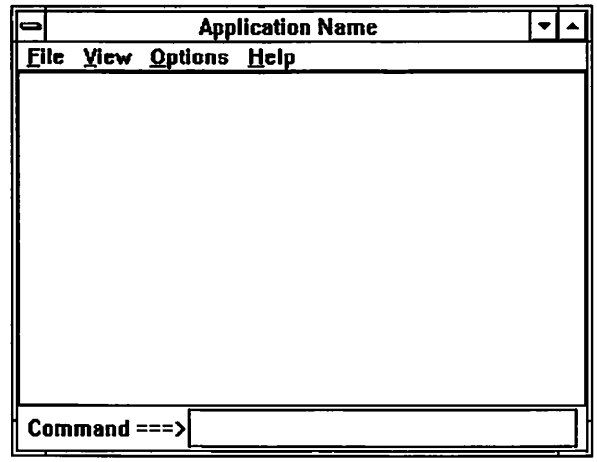


Figure 3-15. Command Area

In Figure 3-15 we did not specify the text of the command prompt, so the Dialog Manager automatically added the text "Command" (or its translated equivalent), which is the default text. If we wanted to specify something other than this text, we could have coded it as tag content, as in this example:

```
<panel name=panel01 depth=15 width=45>Application Name
 <ab>

⋮
 </ab>
  <area>

⋮
  </area>
  <cmdarea>Enter a command
</panel>
```

You can code up to 64 bytes of prompt text when overriding the default text.

The CMDAREA tag also has an optional PMTLOC attribute that allows you to specify the location of the prompt text. The default value for this attribute is BEFORE, which results in the prompt text preceding the command entry field on the same line. The alternate value is ABOVE, which formats the prompt text on the line above the command entry field.

In this markup, we've specified ABOVE as the PMTLOC value for the CMDAREA tag. We also specified prompt text for the command area. Figure 3-16 on page 3-27 shows the formatted result.

```
<panel name=panel01>Application Name
 <ab>

:
 </ab>
  <area>

:
  </area>
  <cmdarea pmtloc=above>Enter a command
</panel>
```

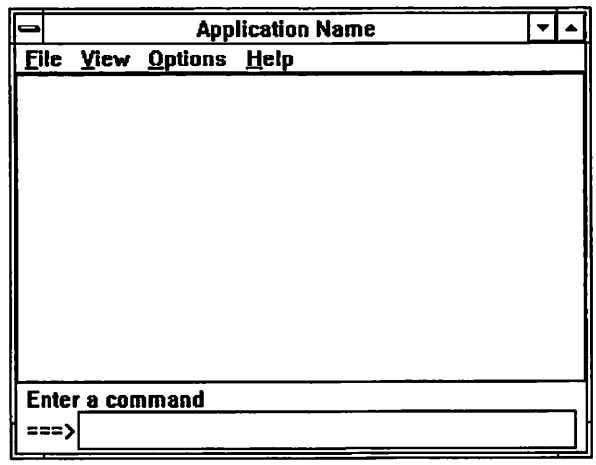


Figure 3-16. Command Area (PMTLOC = ABOVE)

Chapter 4. Variables and Variable Classes

Much of the information displayed within dialog elements is derived directly from the tags used to define it. Other information is obtained dynamically when the application is running, such as:

- Data that the user supplies
- Data that the application supplies
- Data that Dialog Manager supplies.

In all of these cases, the data is derived from values specified in variables.

DTL provides you with tags to *declare* variables and to define the characteristics of these variables using variable classes. Variables and variable classes are considered global because they can be referenced by more than one element within the same source file. All variables referenced by dialog elements must be declared. Variable and variable classes should be used consistently throughout dialog elements that are used in the same application.

Variables declared using DTL are accessible to your application through the Variable Services provided by Dialog Manager. This is how information is communicated between Dialog Manager and your application. Variable Services are described in detail in Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" in the *Dialog Manager Guide and Reference*.

Declaring Variables

You declare variables for dialog elements by coding variable declarations within variable lists. By doing this, you specify whether the variable is located within a variable pool managed by Dialog Manager, or if the variable is managed by a variable access user exit. You also specify the variable name, the variable class of each variable, and, if the variable is an array, the maximum dimension of the array.

The variable list (VARLIST) tag and its required end tag define a variable list. You code variable lists after any variable classes and before any other tags. There are two types of variable lists you can define depending on the optional CLASS attribute associated with the VARLIST tag. If the CLASS attribute is omitted, the variables declared within the variable list are located within a variable pool managed by Dialog Manager. If the CLASS attribute is used, the variables declared within the variable list are managed by the variable access user exit named on the CLASS attribute. For a complete description of user exits, refer to Chapter 13, "Additional Features of the Dialog Manager" in the *Dialog Manager Guide and Reference*.

Although an application can have only one variable list managed by Dialog Manager, you may define multiple variable lists managed by variable access user exits as long as the value of the CLASS attribute for each variable list is unique. Dialog elements can reference variables declared in more than one variable list.

To declare variables, use the VARDCL (variable declaration) tags within a VARLIST definition. The VARDCL tag has two required attributes, NAME and VARCLASS, and one optional attribute, MAXDIM.

NAME The NAME attribute identifies the variable. Dialog elements reference the variable by using the VARDCL NAME value.

For example, a data field would use the variable name on its DATAVAR attribute to specify the variable data associated with its entry field.

VARCLASS The VARCLASS attribute specifies the variable class associated with the variable declaration. Variable classes define the format and length of variable data plus any translations and checks to perform on the data.

MAXDIM The optional attribute, MAXDIM, is used to define the dimension of an array variable. When this attribute is not used, the variable is scalar.

In this example, the first variable list contains two scalar variable declarations referenced by the data fields in the application panel. The second variable list contains an array variable declaration referenced by the list field in the application panel.

```
<!doctype dm system>
:
  <varlist>
    <vardcl name=author varclass=authorc>
    <vardcl name=catnum varclass=catnumc>
  </varlist>
  <varlist class=library>
    <vardcl name=branch varclass=branchc maxdim=50>
  </varlist>

  <panel name=books>Book Title Search
    <area>
      <dtacol pmtwidth=20 entwidth=10 deswidth=20>
        <dtafld datavar=author>Author
        <dtafld datavar=catnum>Catalog number
      </dtacol>
      <lstfld>
        <lstcol datavar=branch>Branch Libraries
      </lstfld>
    </area>
   </panel>
```

Defining Variable Classes

To complete the preceding example, we must code the variable classes that are referenced with the VARDCL VARCLASS attributes. The variable class information must be available so that Dialog Manager knows the maximum amount of data that is acceptable and any other special characteristics that apply.

As stated earlier, when you declare a variable you must specify its associated variable class. The purpose of variable classes is to define the format and length of the variable data. They are also used to define built-in translations and validity checks which are applied to variable values. "Variable Validation" on page 4-9 tells you how to define translations and validity checks.

The VARCLASS (variable class) tag defines a variable class. You code variable classes in the same source file as the dialog elements and variable lists that reference them. Additionally, variable classes must be coded before the dialog elements and variable lists in the source file. You can do this by coding variable classes following the DOCTYPE statement or by coding this information in an external file and embedding the file following the DOCTYPE statement.

There are two required attributes associated with the VARCLASS tag, NAME and TYPE.

NAME The NAME attribute is used to identify and reference the variable class. For example, a variable declaration (VARDCL tag) would use the NAME value as its VARCLASS value to identify the associated variable class.

TYPE The TYPE attribute is used to determine the format and length of the variable data.

In addition to the required attributes described above, the VARCLASS tag has an optional MSG attribute that specifies the message to be displayed if the variable fails the defined validity check. Chapter 8, "Messages" on page 8-1 tells you how to define messages.

Variable Class Types

DTL supports nine unique variable class types which can be logically divided into three groups:

- Character variables
- Time and date variables
- Numeric variables.

Character Variables

The first group includes four types which allow you to specify whether single-byte characters, double-byte characters or both are permitted, as well as the maximum number of bytes the variable can accept. These types are 'CHAR maximum-length', 'DBCS maximum-length', 'ANY maximum-length' and 'MIXED maximum-length', where maximum length defines the number of bytes allowed and is a positive number between 1 and 32767. These types are typically used for character data. Each type is described below:

Type	Description
'CHAR maximum-length'	Specifies a single-byte character string.
'DBCS maximum-length'	Specifies a double-byte character string. The maximum-length must be an even number.
'ANY maximum-length'	Specifies a character string containing single-byte characters, double-byte characters or both. This type can be used to enable your application for both single- and double-byte character systems. This type is treated as CHAR if the system does not support double-byte characters. **Note:** This type is used for OS/2 applications only.

'MIXED maximum-length' Specifies a character string containing single-byte characters, double-byte characters, or both. The maximum number of characters allowed is reduced by two whenever there is a transition from single-byte to double-byte characters.

Note: This type is used for migration to other environments since its behavior is compatible with EBCDIC DBCS. This type is treated as CHAR if the system does not support double-byte characters.

For example, when you define a variable using a VARCLASS of TYPE = 'CHAR 50', the variable accepts up to 50 single-byte characters.

We'll add the variable classes for *authorc* and *branchc* to the markup example on page 4-2. We will assume that an author's last name has a maximum of 40 characters and that each branch library is abbreviated to three characters.

```
<!doctype dm system>

<varclass name=authorc type='char 40'>
<varclass name=branchc type='char 3'>

<varlist>
  <vardcl name=author varclass=authorc>
  <vardcl name=catnum varclass=catnumc>
</varlist>
<varlist class=library>
  <vardcl name=branch varclass=branchc maxdim=50>
</varlist>

<panel name=books>Book Title Search
  <area>
    <dtacol pmtwidth=20 entwidth=10 deswidth=20>
      <dtafld datavar=author>Author
      <dtafld datavar=catnum>Catalog number
    </dtacol>
    <lstfld>
      <lstcol datavar=branch>Branch Libraries
    </lstfld>
  </area>
</panel>
```

Time and Date Variables

The second group of variable classes consists of four types reserved for displaying time and date information. These are ITIME, IDATE, STDTIME and STDDATE. Their internal format differs from their display format. The Dialog Manager automatically converts the internal format to the display format and back again as needed. Dialog Manager also verifies data entered by the user against the format required by the type.

ITIME and STDTIME are used for displaying time information. The internal format for ITIME is expressed as four digits, HHMM, where HH is hours and MM is minutes. The internal format of STDTIME is expressed as six digits, HHMMSS, where HH is hours, MM is minutes and SS is seconds. Note that hours are specified using a 24-hour clock. Before variable values of either of these types are displayed, the internal format is automatically converted to the display format

which includes the national language time separator, denoted by *s*. For ITIME, the display format is HH*s*MM; for STDTIME the display format is HH*s*MM*s*SS.

When a user enters a value for a variable with a type of either ITIME or STDTIME, it must be entered using the display format. It is a good idea to display an explanation of the expected format to the user, so that the value is entered properly. Dialog Manager verifies that the value entered is a valid time, and if no errors are found, the display format is converted to the internal format before the value is stored in the variable pool.

If an error is found, one of three default messages may be issued:

- The first message is used when the time is not correct (if HH is not 00 – 23, MM is not 00 – 59, and SS is not 00 – 59).
- The second message is used when non-numeric characters are entered for the hours, minutes, or seconds portion of the ITIME or STDTIME.
- The third message is used when the national language time separator character is not correct.

You can override these messages by specifying your own message on the MSG attribute.

To retrieve the current time, you can use the system variables ZITIME for ITIME format and ZSTDTIME for STDTIME format. Refer to Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" in the *Dialog Manager Guide and Reference* for more information on system variables.

The following figure shows an example of ITIME and STDTIME internal values and display values.

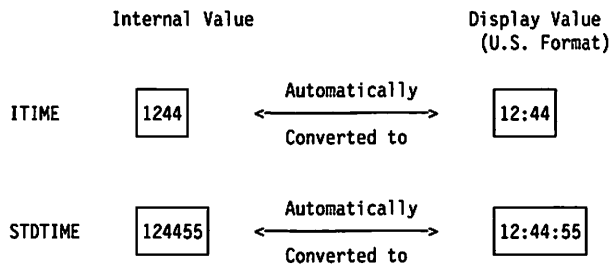


Figure 4-1. ITIME and STDTIME Values

IDATE and STDDATE are used for displaying date information. The internal format for IDATE is expressed as six digits, YYMMDD, where YY is a two-digit year, MM is month, and DD is day; whereas, the internal format for STDDATE is expressed as eight digits, YYYYMMDD, where YYYY is a four-digit year, MM is month, and DD is day. Before variable values of either of these types are displayed, the internal format is automatically converted to the national language date format which includes the national language date separator, denoted by *s*. For IDATE, the display format is MM*s*DD*s*YY (for U.S.); for STDDATE, the display format is MM*s*DD*s*YYYY (for U.S.).

Note: The national language date format is dependent on the configuration of your system.

When a user enters a value for a variable with a type of either IDATE or STDDATE, it must be entered using the national language date format. It is a good idea to display an explanation of the expected format to the user, so that the value is entered properly. Dialog Manager verifies that the value entered is a valid date, and if no errors are found, the national language date format is converted to the internal format before the value is stored in the variable pool. If an error is found, one of three default messages may be issued.

- The first message is used when the date is not correct (if YY is not 00 – 99, or if YYYY is not 0000 – 9999, MM is not 00 – 12, and DD is not 01 – 31, except for months with less than 31 days). Leap year dates are also verified.
- The second message is used when non-numeric characters are entered for the year, month, or day portion of the IDATE or STDDATE.
- The third message is used when the national language date separator character is not correct.

 You can override these messages by specifying your own message with the MSG attribute.

To retrieve the current date, you can use the system variable ZIDATE for IDATE format and ZSTDDATE for STDDATE format. Refer to Chapter 7, "Using Dialog Variables, the Dialog Variable Pool, and Variable Services" in the *Dialog Manager Guide and Reference* for more information on system variables.

The following figure shows an example of IDATE and STDDATE internal values and display values.

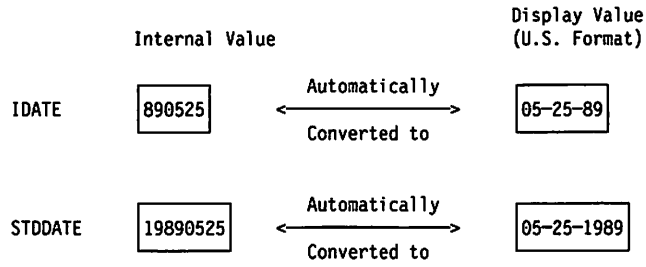


Figure 4-2. IDATE and STDDATE Values

We'll add more variable declarations and variable classes to the example on page 4-4 to show how time and date variable class types are used. We will also add a field for the day the book is due.

```
<!doctype dm system>

<varclass name=authorc type='CHAR 40' msg=liba001>
<varclass name=branchc type='CHAR 3'>
<varclass name=itimec type=ITIME msg=liba002>
<varclass name=stddatec type=STDDATE msg=liba003>
<varclass name=dayc type='CHAR 7'>

<varlist>
  <vardcl name=author varclass=authorc>
  <vardcl name=catnum varclass=catnumc>
  <vardcl name=duetime varclass=itimec>
  <vardcl name=duedate varclass=stddatec>
  <vardcl name=day varclass=dayc>
</varlist>
<varlist class=library>
  <vardcl name=branch varclass=branchc maxdim=50>
</varlist>

<panel name=books>Book Title Search
  <area>
    <dtacol pmtwidth=20 entwidth=10 deswidth=20>
      <dtafld datavar=author>Author's Last Name
      <dtafld datavar=catnum>Catalog number
      <dtafld datavar=duetime>Time book is due
      <dtafld datavar=duedate>Date book is due
      <dtafld datavar=day>Day book is due
    </dtacol>
    <lstfld>
      <lstcol datavar=branch>Branch Libraries
    </lstfld>
  </area>
 </panel>
```

Numeric Variables

The last group of variable classes consists of one type, NUMERIC, used to ensure that a valid number is entered. Numeric is typically used for numbers when mathematical operations are to be performed on them.

When you use this type, you specify the total number of digits and the number of fractional digits allowed in the number. For example, TYPE = 'NUMERIC 10 3' specifies that a number has a total of 10 digits, three of which are fractional. An optional attribute, THS, allows you to reserve enough spaces so the number can be entered with thousands separators. The valid values for THS are YES and NO. A valid number may also contain a sign. In addition, if there are fractional digits, it may also contain a decimal separator. You do not need to reserve a space for the sign or decimal point, the compiler automatically allows space for them.

The internal format of a NUMERIC type is always the same, but the display format may be different based on national language formatting. The Dialog Manager automatically converts the internal NUMERIC format to the display NUMERIC format and back again as needed. The internal format for variables of type NUMERIC includes a preceding minus sign (-) if one is needed, followed by the digits before the decimal separator, a period (.) for the decimal separator, followed by the fractional digits. No thousands separators are present. For example, the internal format for a TYPE = 'NUMERIC 6 2' may be -1234.56.

Before variable values of type NUMERIC are displayed, the internal format is converted to the required display format. Thousands separators are inserted if the optional THS attribute is set (THS=YES). For our previous example, TYPE='NUMERIC 6 2' THS=YES, the display format for U.S. would be -1,234.56.

When the user enters a value for a variable with a type NUMERIC, it must be entered using the required display format. It is a good idea to display an explanation of the expected format to the user, so the value is entered properly. Dialog Manager verifies that the value input is a valid number based on the characteristics used on the TYPE attribute, and if no errors are found, the display format is converted to the internal format before the value is stored in the variable pool. If an error is found, one of five default messages may be issued depending on these errors:

- Incorrect sign
- Incorrect decimal separator
- Incorrect thousands separator(s)
- Non-numeric characters
- Incorrect number of total digits or fractional digits.

You can override these messages by specifying your own message on the MSG attribute.

The following figure shows the internal value *'NUMERIC 6 2'* and a THS value of YES, as well as the resulting display value.

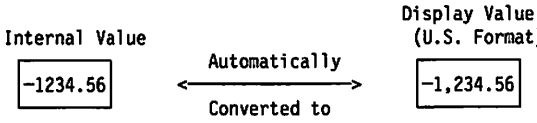


Figure 4-3. NUMERIC Value

We'll add the final variable class, *catnumc*, to the example. We'll assume that each catalog number consists of a total of 10 digits with 2 fractional digits and no thousands separators.

```
<!doctype dm system>

<varclass name=authorc type='CHAR 40' msg=liba001>
<varclass name=catnumc type='NUMERIC 10 2' ths=NO>
<varclass name=branchc type='CHAR 3'>
<varclass name=itimec type=ITIME msg=liba002>
<varclass name=stddatec type=STDDATE msg=liba003>
<varclass name=dayc type='CHAR 7'>

<varlist>
  <vardcl name=author varclass=authorc>
  <vardcl name=catnum varclass=catnumc>
  <vardcl name=duetime varclass=itimec>
  <vardcl name=duedate varclass=stddatec>
  <vardcl name=day varclass=dayc>
</varlist>
<varlist class=library>
  <vardcl name=branch varclass=branchc maxdim=50>
</varlist>

<panel name=books>Book Title Search
  <area>
    <dtacol pmtwidth=20 entwidth=10 deswidth=20>
      <dtafld datavar=author>Author's Last Name
      <dtafld datavar=catnum>Catalog number
      <dtafld datavar=duetime>Time book is due
      <dtafld datavar=duedate>Date book is due
      <dtafld datavar=day>Day book is due
    </dtacol>
    <lstfld>
      <lstcol datavar=branch>Branch Libraries
    </lstfld>
  </area>
</panel>
```

Variable Validation

DTL allows you to define translate lists and validity checks as part of the variable class definition by using tags nested within the VARCLASS tag. These built-in translations and checks are especially useful because Dialog Manager automatically performs them on variable values, so the DM application doesn't need to.

Note: Translations and checks are performed only on variable values that are intended for display. For instance, before displaying the data from a variable specified on the DATAVAR attribute of a DTAFLD tag, the Dialog Manager performs validation (translations and checks) on the variable to obtain the correct display value. However, the Dialog Manager does not perform validation on the variable when it is specified on the CHECKVAR attribute of a CHOICE tag.

Translate Lists

Translate lists provide a means of translating a displayed variable value into a different internal value, and vice versa. Translation can occur on input (when the user enters a value), on output (the value stored in the variable pool is translated before the user sees it), or both. This is based on the USAGE value of the tag that references a variable using a variable class with translate lists.

To associate a translate list within a variable class, code the XLATL (translate list) tag and its required end tag following the VARCLASS start tag.

The type of translate list is determined by the value assigned to the FORMAT attribute of the XLATL tag. The four types of translate lists supported are:

- Upper
- Strip
- Item translation
- User defined.

There is an optional MSG attribute on the XLATL tag that allows you to specify your own message to display when input translation specified by the XLATL fails. If you do not specify a message, Dialog Manager supplies a default message for you. For information about defining your own messages, see Chapter 8, "Messages" on page 8-1.

Upper: Allows you to translate a value to uppercase. To specify this translation, code FORMAT=UPPER on the XLATL tag. Characters are translated to uppercase according to the country code and code page used to compile the source file. This translation is always successful.

Strip: Allows you to remove leading and trailing blanks from a value. To specify this translation, code FORMAT=STRIP on the XLATL tag. This translation is always successful.

We'll add two translate lists to the *authorc* variable class in the example on page 4-9. One translate list will strip leading and trailing blanks and the other will convert the author's name to uppercase.

```
<varclass name=authorc type='CHAR 40' msg=liba001>
  <xlatl format=strip>
  </xlatl>
  <xlatl format=upper>
  </xlatl>
```

The following figure shows the results on input and output translations for the above example.

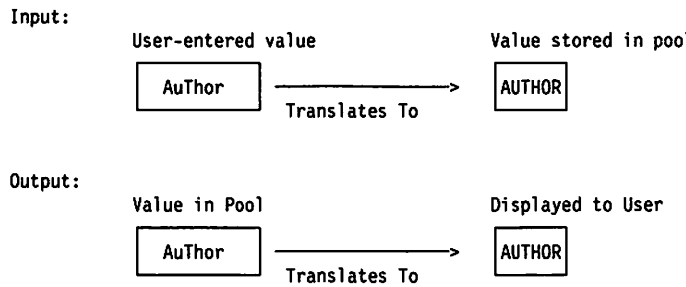


Figure 4-4. Variable Translation Results

Item Translation: Allows you to translate an internal variable value to a displayed value on an item-for-item basis. To specify this translation, either code FORMAT=NONE on the XLATL tag or omit the FORMAT attribute since this is the default. You define the list or lists of possible internal values and the corresponding display values they should be translated to, or from, using the XLATI (translate item) tags nested within the XLATL tag.

To specify an internal value for a translate item, use the VALUE attribute on the XLATI tag.

Note: When the enclosing variable class type is ITIME, IDATE, STDTIME, STDDATE, or NUMERIC, the translate item internal value must adhere to the special internal formats described on pages 4-4 through 4-6.

If no explicit match is found, the data the user enters when these special variable types are used is automatically converted to its internal format before any validity checks are applied.

The display value is the XLATI tag content. If a display value of all blanks or a display value in which trailing blanks are preserved is desired, use the literal (LIT) tag and its required end tag to indicate that blanks are significant.

When a list is searched to find a match, you can specify whether differences in case and whether leading or trailing blanks should be ignored or respected during comparisons. You do this by assigning IGNORE or RESPECT to the CASE and BLANKS attributes on the XLATL tag. IGNORE is the default in both cases.

An explicit match is achieved during translation processing as follows:

- On input, an explicit match occurs when the value the user enters matches one of the specified display values in the translate list. An explicit match also occurs when a display value is omitted (indicating any value is acceptable) and the corresponding internal value is specified.
- On output, an explicit match occurs when the value from the variable pool matches one of the specified internal values in the translate list. An explicit match also occurs when an internal value is omitted (indicating any value is acceptable) and the corresponding display value is specified.

Omitting both the internal value and the display value does not produce an explicit match.

Note: Once an explicit match is found during translation processing, no other translate lists or items are searched and all validity checking is bypassed.

We'll change the variable class *dayc* from the example on page 4-9 to use an internal value for days of the week which is different than the display value. We will also change the variable class for *itimec*, *stddatec* and *catnumc* to show you how the special internal formats used are dependent on the enclosing variable class type.

```
<!doctype dm system>

<varclass name=dayc type='CHAR 1'>
  <xlatl msg=liba004>
    <xlati value=1>Sunday
    <xlati value=2>Monday
    <xlati value=3>Tuesday
    <xlati value=4>Wednesday
    <xlati value=5>Thursday
    <xlati value=6>Friday
    <xlati value=7>Saturday
  </xlatl>
<varclass name=catnumc type='NUMERIC 10 2' THS=NO>
  <xlatl>
    <xlati value=12345678.12>Phone Book
    <xlati value=111.33>Local Newspaper
    <xlati value=44444.23>Magazines
    <xlati>
  </xlatl>
<varclass name=itimec type=ITIMEC>
  <xlatl>
    <xlati value=1200>Noon
    <xlati value=0000>Midnight
    <xlati>
  </xlatl>
<varclass name=stddatec type=STDDATE>
  <xlatl>
    <xlati value=19891231>New Year's Eve
    <xlati value=19900101>New Year's Day
    <xlati>
  </xlatl>
```

The following figure shows how variable values of variable class *dayc* are translated on input and output.

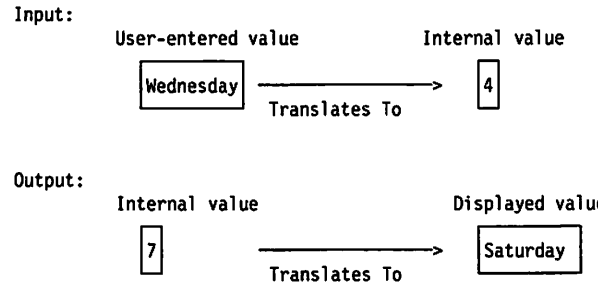


Figure 4-5. Variable Translation

The previous example shows one translate list with a finite number of translation items. This example assumes that the only possible internal values are 1—7 and the only possible display values are the days of the week. A match must be found in this list, or the translation fails.

To allow a non-matching value to be passed on for further processing (either to another translate list or to the validity checks that follow), you code an XLATI tag without an internal or display value to indicate that any value is acceptable. You should keep in mind that if the non-matching value is too big for the space allocated based on the enclosing variable class type, an error is generated.

The following example shows how such an error could occur.

```
<varclass name=dayc type='CHAR 1'>
  <xlatl>
    <xlati value=1>Sunday
    <xlati value=2>Monday
    <xlati value=3>Tuesday
    <xlati value=4>Wednesday
    <xlati value=5>Thursday
    <xlati value=6>Friday
    <xlati value=7>Saturday
    <xlati>
  </xlatl>
```

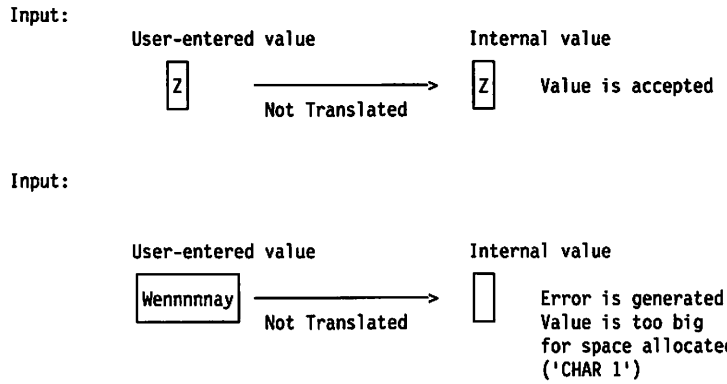


Figure 4-6. Variable Translation

Since multiple translate lists are permitted, we can expand this example to accept either the days of the week spelled out or their accepted abbreviations.

```
<!doctype dm system>

<varclass name=dayc type='CHAR 1'>
  <xlatl>
    <xlati value=1>Sunday
    <xlati value=2>Monday
    <xlati value=3>Tuesday
    <xlati value=4>Wednesday
    <xlati value=5>Thursday
    <xlati value=6>Friday
    <xlati value=7>Saturday
    <xlati>
  </xlatl>
  <xlatl>
    <xlati value=1>Sun
    <xlati value=2>Mon
    <xlati value=3>Tues
    <xlati value=4>Wed
    <xlati value=5>Thur
    <xlati value=6>Fri
    <xlati value=7>Sat
  </xlatl>
```

It is possible to omit only the internal value to indicate that any internal value is acceptable. This affects input and output translate processing differently. When translating on input, the value is not translated before being stored in the variable

pool. When translating on output, any value not already matched is translated to the displayed value.

We will change the *branchc* variable class in the previous example to illustrate translate processing when only the internal value is omitted.

```
<!doctype dm system>

<varclass name=branchc type='CHAR 3'>
  <xlatl format=none case=ignore blanks=ignore>
    <xlati value=1>RAL
    <xlati>CRY
  </xlatl>
```

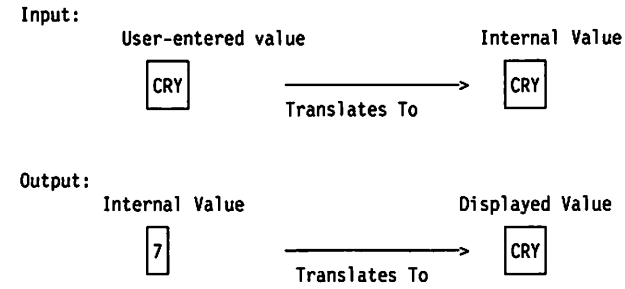


Figure 4-7. Variable Translation

It is also possible to omit only the display value to indicate that any display value is acceptable. This affects input and output translate processing differently. When translating on input, any value not already matched is translated to the internal value. When translating on output, the internal value is not translated before it is displayed.

We will change the *branchc* variable class again to illustrate translate processing when only the display value is omitted.

```
<!doctype dm system>

<varclass name=branchc type='CHAR 3'>
  <xlatl>
    <xlati value=1>RAL
    <xlati value=2>
  </xlatl>
```

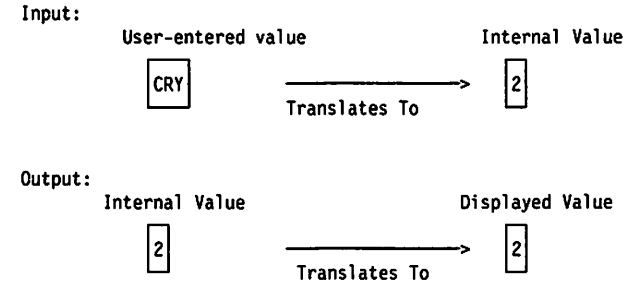


Figure 4-8. Variable Translation

User Defined: Created using a translate user exit. (Chapter 13, "Additional Features of the Dialog Manager" in the *Dialog Manager Guide and Reference* provides a complete description of user exits.) You must specify the name of the user exit using the CLASS attribute. An optional PARM attribute allows you to pass

any specific parameters needed in the translation. The DISPLEN attribute is used to specify the number of bytes needed to convert an internal value to its displayable format. This attribute also ensures that enough space is reserved for the largest possible display value.

Validity Checks

You use validity checks to automatically verify data input by the user. Validity checks are coded after any translate lists, and are bypassed if an explicit match is found on a translate list item.

To associate a validity check within a variable class, code the CHECKL (check list) tag and its required end tag either following the last translate list, or if no translate list exists, following the VARCLASS start tag. The individual check items that define the tests to perform are coded using the CHECKI (check item) tag nested within the check list.

There is an optional MSG attribute on the CHECKL tag that allows you to specify your own message to display when the entered value fails all tests. If you do not specify a message, Dialog Manager supplies a default message for you. For more information about defining your own messages, see Chapter 8, "Messages" on page 8-1.

A value entered by the user must pass only one of the check items defined in a check list for the check list to be considered successful. However, since there can be multiple check lists defined, all check lists must be successful for the validation to be successful.

When the enclosing variable class type is ITIME, IDATE, STDTIME, STDDATE, or NUMERIC, and there is not an explicit match of a translate list, the data the user enters is automatically converted to its internal format before any validity checks are applied. Keep this in mind when you design your validity checks.

The TYPE attribute of the CHECKI tag allows you to specify the various validity tests of the input. Since check items are performed in the order they appear in a check list, they should be coded in order from most likely to succeed to least likely to succeed. Blanks are respected during validation processing, so if you want to remove extraneous blanks, a STRIP translation should be coded prior to the validity checks. You can define the following types of validity checks:

- Range
- Alphabetic
- Character
- Values
- User-defined.

Range: To perform a range test, assign the check item TYPE attribute to RANGE. A range check allows you to check a value within a numeric range including the end points. The PARM1 attribute specifies the lower bound; PARM2 specifies the upper bound. The range delimiters can include 31 digits, and may contain a decimal point and a preceding sign (- or +).

The following example shows how a range check could be added to the VARCLASS *catnum* to ensure that catalog numbers are in the range 50 to 90000000.

```
<!doctype dm system>

<varclass name=catnumc type='NUMERIC 10 2' THS=NO>
  <xlatl>
    <xlati value=12345678.12>Phone Book
    <xlati value=111.33>Local Newspaper
    <xlati value=44444.23>Magazines
    <xlati>
  </xlatl>
  <checkl msg=liba005>
    <checki type=range parm1=50.00 parm2=90000000.00>
  </checkl>
```

In the above example, on output, if the value of the variable is 12345678.12, 111.33, or 44444.23, then it will be translated to "Phone Book," "Newspaper," or "Magazines," respectively. If it is not one of these values, then the display value is the same as the internal value.

On input, if the value entered is "Phone Book," "Newspaper," or "Magazines," it will be translated to 12345678.12, 111.33, or 44444.23, respectively. If any other value is entered, it must be a numeric value between 50 and 90000000.

Alpha: To perform an alphabetic test, assign the check item TYPE attribute to ALPHA. An alpha check limits the characters allowed to A−Z and a−z, and the space character.

We'll add an alpha check to the *authorc* variable class to ensure that authors' names are alphabetic.

```
<!doctype dm system>

<varclass name=authorc type='CHAR 40' msg=liba001>
  <xlatl format=strip>
  </xlatl>
  <xlatl format=upper>
  </xlatl>
  <checkl msg=liba006>
     <checki type=alpha>
  </checkl>
```

Character: To perform a character test, assign the check item TYPE attribute to CHARS. A character tests allows you to specify a list of characters that are either allowed or not allowed in the data the user entered. The PARM1 attribute is assigned EQ (equal) or NE (not equal) to indicate whether the characters are allowed or not allowed. PARM2 specifies the list of characters and is restricted to the set A−Z, a−z, 0−9, the following characters: . , : ; () ' " / - _ & + % * = < > and the space character.

We'll define a check for hexadecimal numbers by specifying the list of characters allowed on input. We'll also add an upper translation, so we only have to code the repeating characters in uppercase. For example, we won't have to worry about all the combinations for A: AA aA Aa aa.

```
<!doctype dm system>

<varclass name=hexc type='char 2'>
  <xlatl format=strip>
  </xlatl>
  <xlatl format=upper>
  </xlatl>
   <checkl msg=liba007>
      <checki type=chars parm1=eq parm2='ABCDEF0123456789'>
   </checkl>
```

Values: To perform a values test, assign the check item TYPE attribute to VALUES. A values test allows you to specify a list of values. The value the user enters must match one or none of the values specified in the list. The PARM1 attribute is assigned EQ or NE to indicate whether the values must match or not. The PARM2 attribute specifies the list of values. Because case is respected in a values check, if case insensitivity is desired, you must code an UPPER translation and code the values all in uppercase. The characters allowed in PARM2 are restricted to the set A−Z, a−z, 0−9, the following characters: . , : ; () ′ ″ / - _ & + % * = < > and the space character.

We'll add another check list to the *hexc* variable class to disallow the values AA, BB, CC, DD, EE, FF.

```
<!doctype dm system>

<varclass name=hexc type='char 2'>
  <xlatl format=strip>
  </xlatl>
  <xlatl format=upper>
  </xlatl>
  <checkl msg=liba007>
     <checki type=chars parm1=eq parm2='ABCDEF0123456789'>
  </checkl>
  <checkl msg=liba008>
     <checki type=values parm1=ne parm2='AA BB CC DD EE FF'>
  </checkl>
```

User Defined: To create your own check, assign the check item TYPE attribute to CLASS. This indicates a check user exit will perform the check at run time. The check user exit named on the CLASS attribute will perform the check. You can also use the optional PARM1 and PARM2 attributes to pass any parameters to the user exit. (Chapter 13, "Additional Features of the Dialog Manager" in the *Dialog Manager Guide and Reference* provides a complete description of user exits.)

Overriding Variable Classes

Some tags, such as DTAFLD, allow you to specify a different variable class for a variable other than the default one that was specified when the variable was declared. This is called an overriding variable class and is used to perform different translates and validity checks from those provided by the default variable class. The overriding variable class and the default variable class TYPE attributes must match.

Variable Class Display Length and Field Size

The variable class display length is the number of characters required to display a variable of a particular variable class considering all possible translations. It can also be thought of as the number of characters a user can enter for a variable of a particular variable class. The variable class display length influences the size of the entry field for data fields, list fields, and selection lists. How the entry field size is actually determined is described in Chapter 5, "Application Panel Fields" on page 5-1. The display length is derived from the following:

- Solely from the variable class TYPE attribute
- Solely from the translate list display values
- From the maximum of the lengths derived from the two above.

The variable class display length is derived solely from the variable class TYPE attribute when no translate lists exist. For types that require a maximum length, this is also the display length (CHAR, ANY, MIXED, and DBCS). For types ITIME, IDATE, STDTIME, STDDATE, and NUMERIC, it is the number of characters required by the display format for each of these types. For example, ITIME requires five characters, HH*s*MM; STDDATE requires ten characters, MM*s*DD*s*YYYY. For type NUMERIC, the display length is the sum of the number of characters required for total digits, decimal separator if there are fractional digits, thousands separators (if THS = YES), and a sign. When translations require an explicit match, the variable class display length is derived solely from the translate lists associated with a VARCLASS. The display length is the number of characters in the longest display value on a translate item.

The following example shows how the variable class display length is determined when there are associated translate lists that require an explicit match. The first variable class has a display length of 5, which is the number of characters in "Three." The second variable class has a display length of 4, which is the number of characters in "Four" and "Five."

```
<varclass name=index type='CHAR 10'>
  <xlatl>
    <xlati value=1000000000>One
    <xlati value=2000000000>Two
    <xlati value=3000000000>Three
  </xlatl>

<varclass name=in type='CHAR 1'>
  <xlatl>
    <xlati value=4>Four
    <xlati value=5>Five
    <xlati value=6>Six
  </xlatl>
```

The display length is the maximum of the display length derived from the variable class TYPE attribute and the translate lists when the translate lists do not require an explicit match.

We'll change the translate lists in the previous example so they don't require a match and show how the display lengths change. The first variable class has a display length of 10, which is the number of characters defined in the TYPE attribute. The second variable class has a display length of 4, which is the number of characters in "Four" and "Five."

```
<varclass name=index type='CHAR 10'>
  <xlatl>
    <xlati value=1000000000>One
    <xlati value=2000000000>Two
    <xlati value=3000000000>Three
    <xlati>
  </xlatl>

<varclass name=index1 type='CHAR 4'>
  <xlatl>
    <xlati value=4>Four
    <xlati value=5>Five
    <xlati value=6>Six
    <xlati>
  </xlatl>
```

Chapter 5. Application Panel Fields

Most of the direct interaction that takes place between the user and the application is through the use of interactive fields. They provide a means for the user to communicate data to the application, as well as receive data from the application.

The type of interaction the user has with the application depends on the task. This, in turn, determines the fields' characteristics. The appearance of the fields, the application's response to user input, and assistance such as messages and help information must all be considered when defining an interactive field.

In this chapter, we tell you how to use the Dialog Tag Language to define the following types of fields and their operating characteristics:

- Data fields
- Selection fields
- Selection lists
- List fields.

This chapter also describes some features that are common to more than one field, such as field prompts.

Field Prompts

A field prompt is static, descriptive text that explains the field with which it is associated. Data fields and selection fields support the use of field prompts. To define a field prompt for a data field or selection field, specify the prompt text as the tag text on the DTAFLD and SELFLD tags.

The PMTLOC attribute defines the location of the prompt using one of the following values:

PMTLOC = ABOVE
The prompt is drawn above and left-aligned with the field. This is the default for selection fields.

PMTLOC = BEFORE
The prompt is drawn directly in front of and on the same line as the field. This is the default for data fields.

You can also define the amount of space the prompt uses, by specifying the PMTWIDTH attribute. If the prompt text is longer than the width you specify on PMTWIDTH, the prompt is word-wrapped on multiple lines. Using the PMTWIDTH attribute can ensure that multiple fields with prompts are aligned evenly.

Figure 5-1 shows the prompt locations.

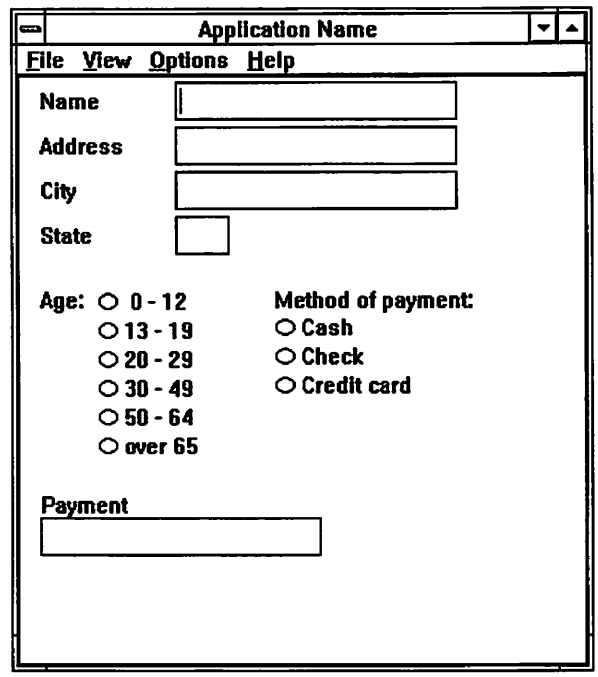


Figure 5-1. Prompt Locations

The **Name**, **Address**, **City**, and **State** data fields show the prompts in front of the fields (PMTLOC=BEFORE), as does the **Age** field, which shows a prompt for a selection field. The same prompt width is used on the first four data fields so that they align evenly. The **Method of payment** and **Payment** fields demonstrate the prompt above the field (PMTLOC=ABOVE).

Here is the markup used to demonstrate the field prompts in the Figure 5-1 on page 5-2:

```
<!doctype dm system>

<varclass name=sampcls type ='char 20'>
<varclass name=statcls type='char 2'>

<varlist>
  <vardcl name=name varclass=sampcls>
  <vardcl name=addr varclass=sampcls>
  <vardcl name=city varclass=sampcls>
  <vardcl name=stat varclass=statcls>
  <vardcl name=pay varclass=sampcls>
</varlist>

<panel name=pmt01>Application Name
  <ab>
⋮
  </ab>
  <area>
    <dtafld datavar=name pmtwidth=10>Name
    <dtafld datavar=addr pmtwidth=10>Address
    <dtafld datavar=city pmtwidth=10>City
    <dtafld datavar=stat pmtwidth=10>State
    <divider>
      <region dir=horiz>
        <selfld pmtloc=before>Age:
          <choice> 0 - 12
          <choice>13 - 19
          <choice>20 - 29
          <choice>30 - 49
          <choice>50 - 64
          <choice>over 65
        </selfld>
        <divider gutter=4>
        <selfld>Method of payment:
          <choice>Cash
          <choice>Check
          <choice>Credit card
        </selfld>
      </region>
    <divider>
    <dtafld datavar=pay pmtloc=above>Payment
  </area>
</panel>
```

Figure 5-2 shows how the prompt width affects the appearance of the prompt text.

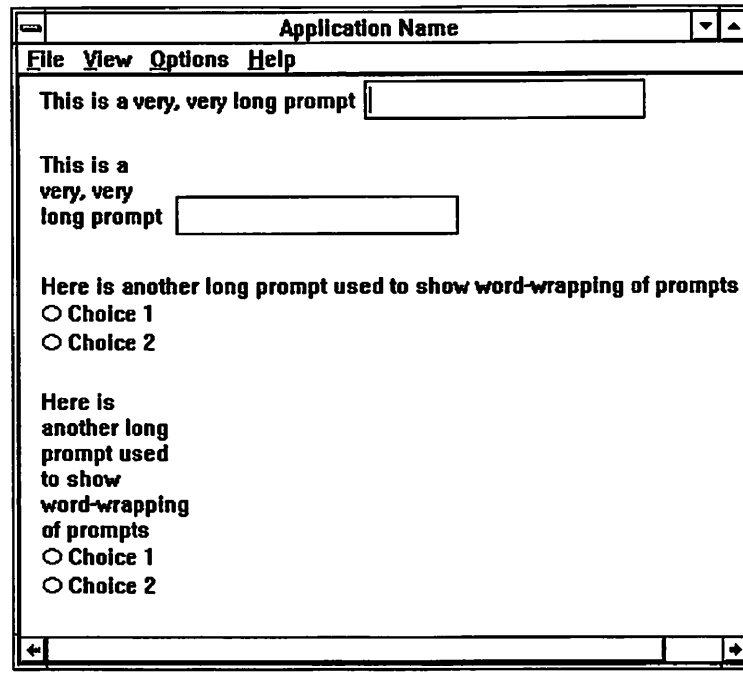


Figure 5-2. Prompt Widths

The prompts in the two data fields are formatted differently. The prompt text of the first data field is not wrapped. It formats on one line, using as much space as necessary (up to a maximum width of 128 characters). The second data field has the same prompt text, with a prompt width that is less than the amount of space needed, so the prompt text is wrapped to another line. Similarly, the two selection fields also demonstrate how the prompt text appears based on the prompt width. The prompt texts of data fields and selection fields can be displayed differently by omitting or specifying different values for the PMTWIDTH value.

The following is the markup used to demonstrate the field prompts in Figure 5-2 on page 5-4:

```
<!doctype dm system>

<varclass name=sampcls type='char 10'>

<varlist>
  <vardcl name=sample varclass=sampcls>
</varlist>

<panel name=pmt02>Application Name
  <ab>
⋮
  </ab>
  <area>
    <dtafld datavar=sample>This is a very, very long prompt
    <divider>
    <dtafld datavar=sample pmtwidth=10>This is a very, very long prompt
    <divider>
    <selfld>Here is another long prompt used to show
            word-wrapping of prompts
      <choice>Choice 1
      <choice>Choice 2
    </selfld>
    <divider>
    <selfld pmtwidth=12>Here is another long prompt used to show
                        word-wrapping of prompts
      <choice>Choice 1
      <choice>Choice 2
    </selfld>
  </area>
</panel>
```

Defining Data Fields

Data fields are used to display variable data and to allow the user to enter data. To define a data field, use the DTAFLD tag. Every data field must have an associated variable, which is specified on the required DATAVAR attribute. Like all variables used on the panel, the variable named on the DATAVAR attribute must be declared using the VARDCL tag.

The purpose of the data field is defined using one of these values on the USAGE attribute of the DTAFLD tag:

IN Defines an entry (input-only) data field. An entry data field allows the user to enter data. When an entry field is initially displayed, it is blank.

OUT Defines an output-only data field. An output-only data field is used to display the current value of the variable associated with the data field. The cursor cannot be placed in an output-only data field; therefore, the user cannot interact with an output-only field.

BOTH Defines an input/output data field. When an input/output field is initially displayed, the current value of the associated variable is displayed, and the user can enter data into the field as well. If you do not specify the USAGE attribute, BOTH is the default.

Data fields support field prompts, which can be placed in front of or above the data field.

This panel contains examples of all three types of data fields:

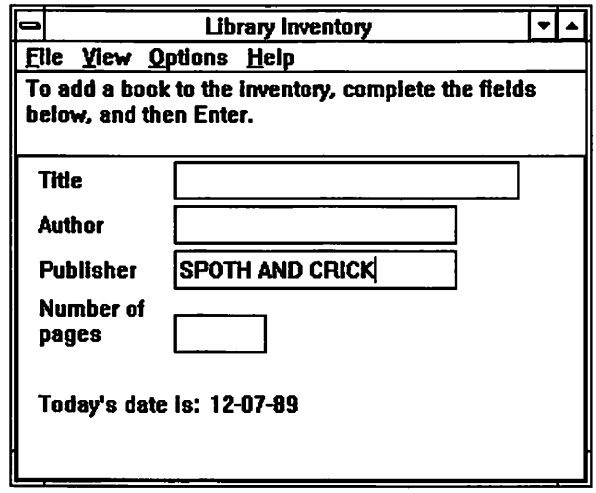


Figure 5-3. Data Fields

Here's the markup for Figure 5-3:

```
<!doctype dm system>

<varclass name=titlcls type='char 60'>
<varclass name=bookcls type='char 20'>
<varclass name=pagecls type='char 5'>
<varclass name=datecls type=idate>

<varlist>
  <vardcl name=title   varclass=titlcls>
  <vardcl name=author  varclass=bookcls>
  <vardcl name=publish varclass=bookcls>
  <vardcl name=pages   varclass=pagecls>
  <vardcl name=zidate  varclass=datecls>
</varlist>

<panel name=dfdxmpl>Library Inventory
  <ab>
:
  </ab>
  <topinst>To add a book to the inventory, complete the fields below,
           and then Enter.
  <area>
    <dtafld datavar=title usage=in pmtwidth=10>Title
    <dtafld datavar=author usage=in pmtwidth=10>Author
    <dtafld datavar=publish pmtwidth=10>Publisher
    <dtafld datavar=pages usage=in pmtwidth=10>Number of pages
    <divider type=solid>
    <dtafld datavar=zidate usage=out>Today's date is:
  </area>
</panel>
```

In the previous example, there are three entry data fields. The value of the associated variable is not displayed in an entry data field, so when the panel is initially displayed, the **Title**, **Author**, and **Number of pages** fields are blank. The **Publisher** data field assumes the default usage, BOTH, so the current value of the associated variable, *publish*, is displayed in the data field when the panel is initially displayed. There is one output-only data field, and it is used to display the current date. The user cannot interact with this data field, since it is used only to display variable data. The user can type into any of the data fields except output-only data fields, as indicated by the boxes around the data fields.

Data Field Width

The width of the data field is determined by the amount of data that can be entered into the data field (the display length specified by the variable class), in conjunction with the ENTWIDTH attribute. When the ENTWIDTH attribute is not used, the width of the data field is determined using these criteria:

- If the number of characters specified on the variable class is less than or equal to 25, the width of the data field is the number of character units specified on the variable class.
- If the number of characters specified on the variable class is greater than 25, the width of the data field is 25 character units.

For example, in the markup on page 5-6, any variable that belongs to the *bookcls* can contain 20 bytes of data. The data fields that have a DATAVAR belonging to the *bookcls* (**Author** and **Publisher**) have a width of 20 character units. Any variable that belongs to the *titlcls* can contain 60 bytes of data, but the width of the **Title** data field (whose data variable belongs to *titlcls*) is only 25 character units wide. The user can still enter 60 bytes into the **Title** field. In this case, the data field is not as wide as the amount of data that it can contain, so it is horizontally scrollable.

When word-wrapping takes place on output fields, the wrapping is based on the current data value. The current value of the data field is word-wrapped on more than one line if necessary. This occurs if a variable is defined at 60 characters and the variable is currently using 10 characters. If the field width is defined at 20 characters, wrapping does not occur because the data is 10 characters, not the maximum length of 60.

The ENTWIDTH attribute of the DTAFLD tag can be used to make the data field smaller. For input/both data fields, the box around the entry field is the width of the number of characters specified on ENTWIDTH, but the user is still able to enter the number of characters as specified on the variable class. On output-only data fields, the value of the data field is displayed within the specified ENTWIDTH.

The ENTWIDTH value can also be greater than the width of the data field. In this case, the data field is left-justified within the space defined by the ENTWIDTH attribute.

You can also use ENTWIDTH to line up multiple data field descriptions.

The following example modifies the previous example to show the ENTWIDTH attribute added to the **Author** data field.

```
<!doctype dm system>

<varclass name=titlcls type='char 60'>
<varclass name=bookcls type='char 20'>
<varclass name=pagecls type='char 5'>
<varclass name=datecls type=idate>

<varlist>
  <vardcl name=title   varclass=titlcls>
  <vardcl name=author  varclass=bookcls>
  <vardcl name=publish varclass=bookcls>
  <vardcl name=pages   varclass=pagecls>
  <vardcl name=zidate  varclass=datecls>
</varlist>

<panel name=dfdxmp2>Library Inventory
  <ab>
:
  </ab>
  <topinst>To add a book to the inventory, complete the fields below,
  and then Enter.
  <area>
    <dtafld datavar=title usage=in pmtwidth=10>Title
    <dtafld datavar=author usage=in pmtwidth=10 entwidth=15>Author
    <dtafld datavar=publish pmtwidth=10>Publisher
    <dtafld datavar=pages usage=in pmtwidth=10>Number of pages
    <divider type=solid>
    <dtafld datavar=zidate usage=out>Today's date is:
  </area>
</panel>
```

Here is the screen generated by this markup:

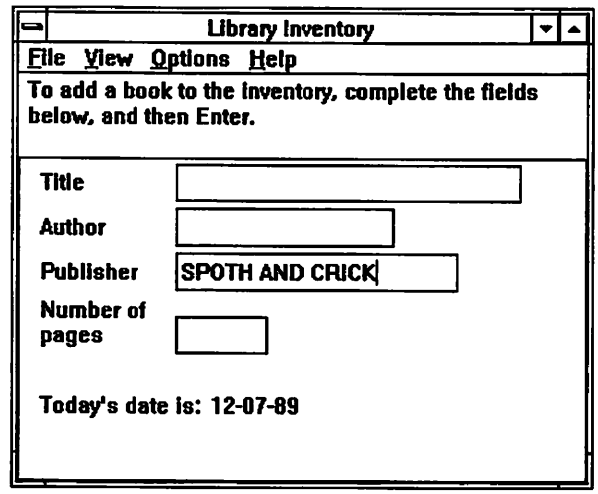


Figure 5-4. Data Field Entry Width

Notice that the **Author** data field is now 15 character units wide because of the ENTWIDTH value of the DTAFLD tag, but the user can still enter 20 bytes of data as specified in the TYPE attribute of the associated VARCLASS tag. Because the **Author** data field is not as wide as the amount of data that it can contain, it is horizontally scrollable.

This panel changes the panel in Figure 5-4 to contain output-only data fields, and the ENTWIDTH attribute is moved from the **Author** field to the **Title** data field.

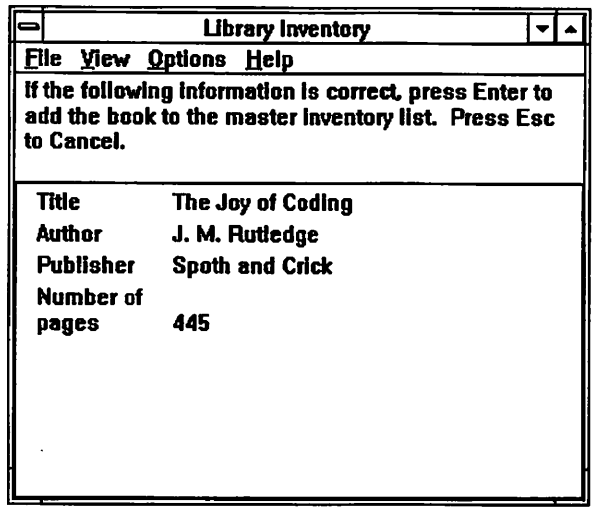


Figure 5-5. Output Data Field

Here is the markup for the panel in Figure 5-5:

```
<!doctype dm system>

<varclass name=titlcls type='char 60'>
<varclass name=bookcls type='char 20'>
<varclass name=pagecls type='char 5'>

<varlist>
  <vardcl name=title   varclass=titlcls>
  <vardcl name=author  varclass=bookcls>
  <vardcl name=publish varclass=bookcls>
  <vardcl name=pages   varclass=pagecls>
</varlist>

<panel name=dfdxmp3>Library Inventory
  <ab>
:
  </ab>
  <topinst>If the following information is correct, press Enter to add
           the book to the master inventory list.  Press Esc to Cancel.
  <area>
    <dtafld datavar=title usage=out pmtwidth=10 entwidth=15>Title
    <dtafld datavar=author usage=out pmtwidth=10>Author
    <dtafld datavar=publish usage=out pmtwidth=10>Publisher
    <dtafld datavar=pages usage=out pmtwidth=10>Number of pages
  </area>
</panel>
```

Data Field Descriptions

In addition to a field prompt, you can provide additional descriptive text for a data field using the the DTAFLDD (data field description) tag. You code the DTAFLDD tag following the definition of the data field being described. The DTAFLDD tag has no attributes or required end tag. Multiple data field descriptions can be coded if necessary, and each description begins a new line.

The data field description is drawn on one line taking up as much room as necessary (up to a maximum width of 128 characters), unless you have used the DESWIDTH attribute on the DTAFLD tag. If the DESWIDTH attribute is defined, the data field description is displayed within the description width specified, and word-wrapped on multiple lines, if necessary.

This panel adds data field descriptions to the data fields from a previous panel.

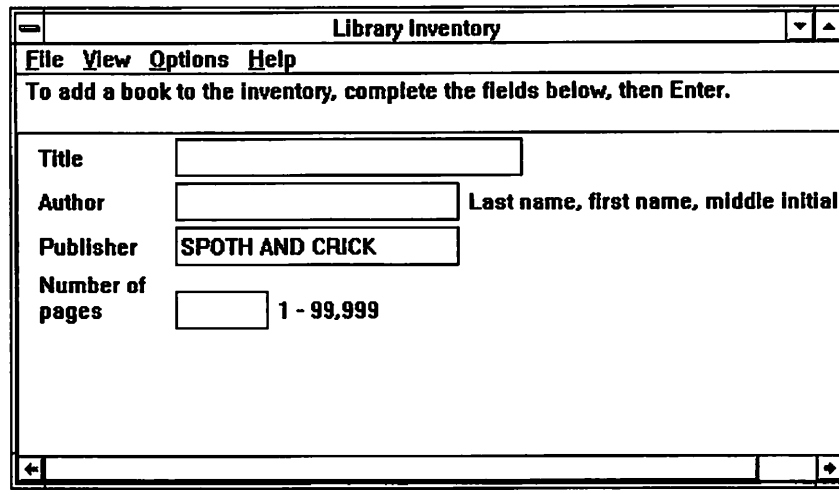


Figure 5-6. Data Field Description

Here is the markup used to generate the panel in Figure 5-6 on page 5-10:

```
<!doctype dm system>

<varclass name=titlcls type='char 60'>
<varclass name=bookcls type='char 20'>
<varclass name=pagecls type='char 5'>

<varlist>
  <vardcl name=title   varclass=titlcls>
  <vardcl name=author  varclass=bookcls>
  <vardcl name=publish varclass=bookcls>
  <vardcl name=pages   varclass=pagecls>
</varlist>

<panel name=dfdxmp4>Library Inventory
  <ab>
⋮
  </ab>
  <topinst>To add a book to the inventory, complete the fields below,
  then Enter.
  <area>
    <dtafld datavar=title usage=in pmtwidth=10>Title
    <dtafld datavar=author usage=in pmtwidth=10>Author
      <dtafldd>Last name, first name, middle initial
    <dtafld datavar=publish pmtwidth=10>Publisher
    <dtafld datavar=pages usage=in pmtwidth=10>Number of pages
      <dtafldd>1 - 99,999
  </area>
</panel>
```

Data Field Help

The Dialog Manager allows you to provide help on a data field using the HELP attribute on the DTAFLD tag. If you specify the name of the help panel for the data field, the Dialog Manager knows which help panel to display when the user selects help on the data field. If you do not specify help for a data field, the extended help panel is displayed.

The following example shows how you provide help for data fields.

```
<!doctype dm system>

<varclass name=titlcls type='char 60'>
<varclass name=bookcls type='char 20'>
<varclass name=pagecls type='char 5'>

<varlist>
  <vardcl name=title   varclass=titlcls>
  <vardcl name=author  varclass=bookcls>
  <vardcl name=publish varclass=bookcls>
  <vardcl name=pages   varclass=pagecls>
</varlist>

<panel name=dfdxmp5>Library Inventory
  <ab>
:
  </ab>
  <topinst>To add a book to the inventory, complete the fields below,
  then Enter.
  <area>
    <dtafld datavar=title help=hlptitl pmtwidth=10>Title
    <dtafld datavar=author help=hlpauth pmtwidth=10>Author
    <dtafld datavar=publish help=hlppubl pmtwidth=10>Publisher
    <dtafld datavar=pages help=hlppage pmtwidth=10>Number of pages
   </area>
</panel>
```

Note: Because output-only data fields cannot contain the cursor, the user cannot select help on them. You should ensure that the help information for an output-only data field is included in the extended help information for the panel.

Other Data Field Attributes

There are several other attributes you can specify to tailor a data field to meet the requirements of your application. The following list describes each of the remaining DTAFLD attributes, and what you can do with them:

REQUIRED

This attribute allows you to indicate if the data field requires input. When you assign a value of *yes* to this attribute, the user must enter data into the field before the Dialog Manager will accept the panel as valid. The default REQUIRED value is NO. This attribute is only valid for input/both data fields.

MSG

This attribute identifies the message that should be displayed when the user does not enter any data into an input-required data field. If you do not specify this attribute, the Dialog Manager displays a default message. This attribute is valid only if REQUIRED=YES.

Chapter 8, "Messages" on page 8-1 tells you how to define application messages.

ALIGN

This attribute allows you to align the variable data within the data field. The default value for ALIGN is *start*, which aligns the data from the left side of the data field. You can also center the data within the field with the *center* value, or justify the data from the right side of the field with the *end* value.

AUTOCLR
This attribute allows you to specify an auto-clear data field. On an auto-clear data field, the first character a user enters into the data field causes the remaining contents of the data field to be erased. The default value for AUTOCLR is NO. This attribute is only valid for input/both data fields.

DISPLAY
The value you assign to this attribute, either *yes* (the default) or *no*, determines if the data appears on the screen when the user enters it. A good way to use DISPLAY=NO is for defining a password. This attribute is only valid for input/both data fields.

VARCLASS
This attribute allows you to override the variable class that is specified on the variable declaration (VARDCL) for the data field's data variable (DATAVAR). See Chapter 4, "Variables and Variable Classes" on page 4-1 for a description of variables and variable classes.

Defining Selection Fields

Selection fields allow the user to select from a group of choices on an application panel. You can specify if only one choice can be selected from a selection field, more than one choice, or if an immediate response takes place when the user makes a selection.

In all of the above cases, you use the same DTL tags to define a selection field. The SELFLD (selection field) tag and its required end tag define a selection field. The CHOICE (selection choice) tag defines a choice within a selection field. You code the CHOICE tags between the SELFLD start and end tags, like this:

```
<selfld>
  <choice>
  <choice>
  <choice>
</selfld>
```

Each CHOICE tag defines a choice within the selection field.

Like data fields, selection fields support field prompts which can be placed in front of or above the selection field. Field prompts are described in "Field Prompts" on page 5-1.

To define the selection field type (single-choice, multiple-choice, or immediate-action) you use the TYPE attribute associated with the SELFLD tag. The values you can assign to TYPE are:

SINGLE Specifies the selection field as being a single-choice field. Choices in a single-choice selection field appear as radio buttons. Single-choice selection fields use *implicit selection*. In implicit selection, choices are automatically selected when the cursor is moved to the choice. The old choice, if one exists, is de-selected. This is the default value.

MULTI Specifies the selection field as being a multiple-choice field. Choices in a multiple-choice selection field appear as check boxes.

ACTION Specifies that an immediate action occurs when the user selects a choice. Choices in an immediate-action selection field appear as pushbuttons.

The CHOICE tag has two attributes associated with it that are very important when defining a selection field, CHECKVAR and MATCH. In all three types of selection fields, the CHECKVAR and MATCH attributes are used to preselect choices in the selection field. For single- and multiple-choice selection fields, these attributes also communicate to the application which selections were made by the user.

The value specified on the CHECKVAR attribute is the name of a dialog variable that is defined by the application. Both the application and the Dialog Manager can set the check variable. The following sections describe how the CHECKVAR and MATCH attributes are used for each specific type of selection field.

Single-choice Fields

Use a single-choice selection field (radio buttons) when you have a fixed set of choices that are mutually exclusive. That is, the user can select only one of the choices, and selecting a choice de-selects the previously selected choice. You should specify the default (preselected) choice in a single-choice selection field so that one item is already selected when the panel is displayed.

In order to preselect choices in a single-choice selection field, and to find out which choice was selected by the user, you should specify the CHECKVAR and MATCH attributes for each CHOICE tag. For a single-choice field, all of the enclosed choices should reference the same check variable, but they should have unique MATCH values. The following markup shows how this is coded.

```
<!doctype dm system>

<varclass name=daycls type ='char 1'>

<varlist>
  <vardcl name=day varclass=daycls>
</varlist>

<panel name=singsel>Schedule Appointments
  <ab>

:
  </ab>
  <topinst>Choose the most convenient day for your appointment,
           then Enter.
  <area>
    <selfld>Weekdays:
      <choice checkvar=day match=M>Monday
      <choice checkvar=day match=T>Tuesday
      <choice checkvar=day match=W>Wednesday
      <choice checkvar=day match=H>Thursday
      <choice checkvar=day match=F>Friday
    </selfld>
  </area>
</panel>
```

If we want to preselect a certain choice, we would set the check variable, *day*, to the match value for that choice. Assume that the check variable, *day*, is set to *M* before the panel is displayed. When the panel is displayed, the choice, **Monday** will be selected as shown in Figure 5-7 on page 5-15.

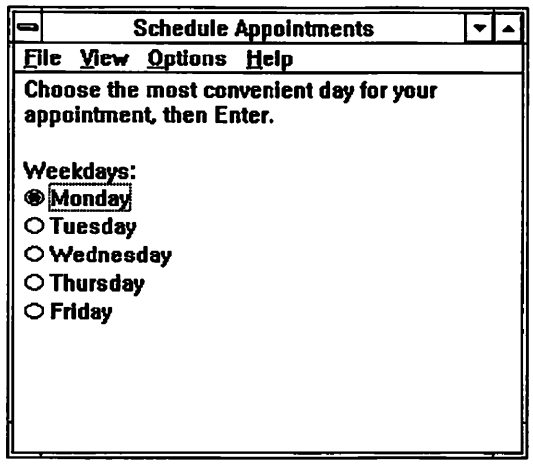


Figure 5-7. Single-Choice Selection Field

If the user decides that another day is more convenient, another choice might be selected. This causes the check variable to be updated with the match value of the newly selected choice. For example, if the user selects **Friday**, the check variable, *day*, will contain "F" when control is returned to the application.

Note: The TYPE attribute does not have to be specified on a single-choice selection field because TYPE=SINGLE is the default.

Multiple-choice Fields

Use a multiple-choice selection field (check boxes) when you have several choices for the user, but they are not mutually exclusive. Each choice acts independently as a toggle, and selecting one of the choices does not affect any of the other choices in the selection field.

In order to preselect choices in a multiple-choice selection field, and to find out which choices were selected by the user, specify the CHECKVAR and MATCH attributes for each CHOICE tag.

On a multiple-choice selection field, define a unique check variable for each enclosed CHOICE, and let the MATCH value default to 1. Here's how a multiple-choice selection field is coded:

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>

<varlist>
  <vardcl name=dry varclass=sampcls>
  <vardcl name=cut varclass=sampcls>
  <vardcl name=per varclass=sampcls>
  <vardcl name=fac varclass=sampcls>
  <vardcl name=man varclass=sampcls>
  <vardcl name=ped varclass=sampcls>
</varlist>

<panel name=multsel>Schedule Appointments
  <ab>

:
  </ab>
  <area>
    <selfld type=multi>Choose the services needed, then Enter.
      <choice checkvar=dry>Dry haircut
      <choice checkvar=cut>Shampoo, haircut, and style
      <choice checkvar=per>Permanent or body wave
      <choice checkvar=fac>Facial
      <choice checkvar=man>Manicure
      <choice checkvar=ped>Pedicure
    </selfld>
  </area>
</panel>
```

You specify default choices for a multiple choice selection field just as you would for a single-choice selection field. Set the check variable for the preselected choices to the match values for those choices.

When the user selects a choice in a multiple-choice selection field, the Dialog Manager toggles the choice as follows:

- If the choice is already selected, the Dialog Manager de-selects the choice, and sets the check variable to 0.
- If the choice is not selected, the Dialog Manager selects the choice, and sets the check variable to the MATCH value for the choice, or 1 if the MATCH attribute is not specified.

In the preceding markup, the MATCH attribute was not specified, so the check variables will be toggled between 0 and 1 (the default MATCH value) as the user selects and de-selects items.

Figure 5-8 on page 5-17 shows how a multiple-choice selection field appears.

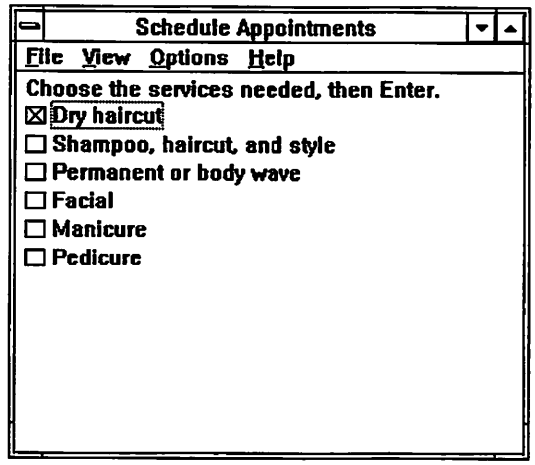


Figure 5-8. Multiple-Choice Selection Field

Immediate-action Fields

Immediate-action selection fields (pushbuttons), like action bars, are used to indicate immediate actions to the user. Typically, action bars are used in primary windows, and action selection fields are used in pop-up windows. To ensure that an immediate action will occur when the choice is selected, you should code an ACTION tag, with the RUN attribute, under each CHOICE tag.

The CHECKVAR and MATCH attributes on immediate-action selection fields are used differently than in the other types of selection fields. On immediate-action selection fields, these attributes are used to define a default action on your panel. A pushbutton that represents a default action is drawn with a heavy border as a visual cue to the user to show which action will occur when Enter is pressed.

To define a default action, specify the CHECKVAR and MATCH attributes on the choice that represents the default action, and set the check variable to the MATCH value before the panel is displayed. You should only specify one default action on a given panel.

The following markup shows how an immediate-action selection field with a default action is coded.

```
<!doctype dm system>

<varclass name=actcls type ='char 3'>

<varlist>
  <vardcl name=daction varclass=actcls>
</varlist>

<panel name=actsel>
  <selfld type=action dir=horiz>
    <choice checkvar=daction match=OK>Ok
      <action run=ENTER>
    <choice>Cancel
      <action run=CANCEL>
    <choice>Help
      <action run=EXHELP>
  </selfld>
</panel>
```

If the check variable, *daction*, is set to the MATCH value, *OK*, the default pushbutton will appear with a heavy border, as seen in Figure 5-9.

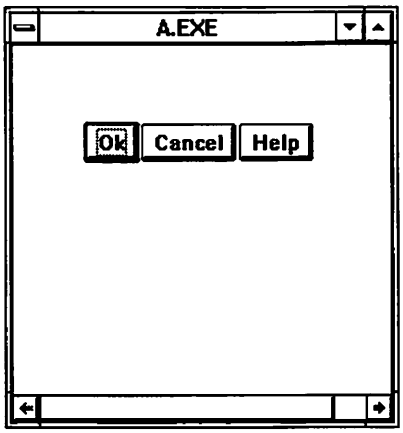


Figure 5-9. Immediate-Action Selection Field

The preceding example also demonstrates the DIR attribute of the SELFLD tag, which allows you to specify the direction in which the choices are arranged. The default arrangement of the choices in a selection field is vertical, but you can specify DIR=HORIZ so that the choices are horizontally arranged. The horizontal arrangement is recommended by Common User Access for the row of pushbuttons along the bottom of a panel within a pop-up window.

Selection Field Choice Actions

In addition to the RUN action, you can specify other types of actions to occur when a selection field choice is selected. The SETVAR and TOGVAR attributes on the ACTION tag can be used to set and toggle variables. This is useful when the variable being changed is shared with another field on the panel. A good example of this is given in "Unavailable Choices" on page 5-20.

Note: Remember that any SETVAR or TOGVAR actions must be coded before the RUN action, since RUN is an ending action.

Mnemonics

Each selection field choice can have an associated mnemonic character defined with the M tag. When you add an M tag to the choice text, it designates the character that immediately follows it as the mnemonic for that choice, and underscores it on the user's display. The user can then select that choice from anywhere within that selection field by entering the mnemonic.

Note: In order to provide a consistent user interface, you must specify a mnemonic for every choice in a selection field or specify no mnemonics for the selection field at all. For this reason, if you specify mnemonics, each mnemonic character within a selection field definition must be unique.

The following figure shows how mnemonics look.

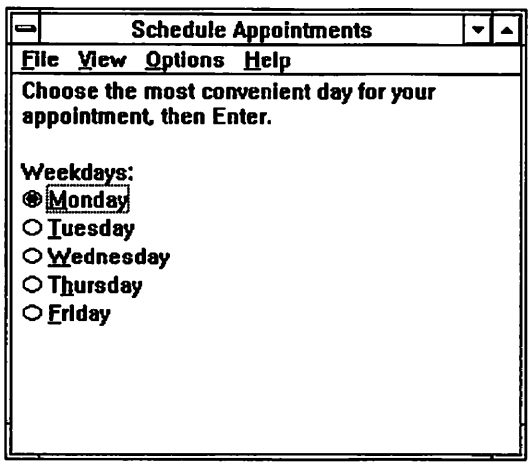


Figure 5-10. Selection Field Mnemonics

Here is the markup used to produce the panel in Figure 5-10:

```
<panel name=singsel>Schedule Appointments
  <ab>

:
  </ab>
  <topinst>Choose the most convenient day for your appointment, then Enter.
  <selfld>Weekdays:
    <choice><m>Monday
    <choice><m>Tuesday
    <choice><m>Wednesday
    <choice>T<m>hursday
    <choice><m>Friday
  </selfld>
</panel>
```

Selection Field Help

The Dialog Manager allows you to provide help on each selection field choice, and the selection field as a whole, by using the HELP attribute of the SELFLD and CHOICE tags. If you specify the name of the help panel for the selection field or choice, the Dialog Manager knows which help panel to display when the user selects help on that choice. If you do not specify help for a selection field choice, the help for the selection field is displayed when the user selects help. If there is no help defined for the selection field, the extended help panel is displayed.

The following example shows how you code help for the selection field.

```
<selfld help=dayhelp>Weekdays:
  <choice checkvar=day match=M >Monday
  <choice checkvar=day match=T >Tuesday
  <choice checkvar=day match=W >Wednesday
  <choice checkvar=day match=H >Thursday
  <choice checkvar=day match=F >Friday
</selfld>
```

This example shows how to code help for each individual choice:

```
<selfld type=multi>Choose the services needed:
  <choice help=dryhlp  >Dry haircut
  <choice help=cuthlp  >Shampoo, haircut, and style
  <choice help=permhlp >Permanent or body wave
  <choice help=facehlp >Facial
  <choice help=manihlp >Manicure
  <choice help=pedihlp >Pedicure
</selfld>
```

Unavailable Choices

Unavailable emphasis is a visual cue to the user that a particular choice cannot be selected because of a condition that exists in the application. The Dialog Manager allows you to show unavailable emphasis for a selection field choice by using the AVAILVAR attribute of the CHOICE tag.

The attribute value for the AVAILVAR tag is the name of a variable that the Dialog Manager evaluates at run time to determine the availability of a selection field choice. If the variable is set to true (1), then the choice is displayed as an available choice and can be selected by the user. If the variable is set to false (0), then the choice is displayed with unavailable emphasis (grayed), and cannot be selected by the user. If you do not use the AVAILVAR attribute, the choice is displayed as available.

Although an unavailable choice cannot be selected, the user can move the cursor to the choice so that help can be selected. If the user attempts to select an unavailable choice, an alarm is sounded as audible feedback that the choice is unavailable.

For example, let's continue the appointment scheduling application used previously. If the user has already scheduled enough appointments on Wednesday, you should visually indicate this by de-emphasizing the **Wednesday** radio button, as shown in Figure 5-11 on page 5-21.

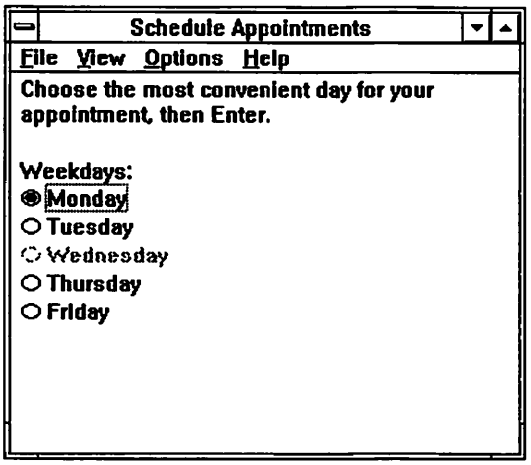


Figure 5-11. Unavailable Selection Field Choices

Here is the markup:

```
<!doctype dm system>

<varclass name=daycls type ='char 1'>

<varlist>
  <vardcl name=day   varclass=daycls>
  <vardcl name=fullM varclass=daycls>
  <vardcl name=fullT varclass=daycls>
  <vardcl name=fullW varclass=daycls>
  <vardcl name=fullH varclass=daycls>
  <vardcl name=fullF varclass=daycls>
</varlist>

<panel name=singsel>Schedule Appointments
  <ab>

:
  </ab>
  <topinst>Choose the most convenient day for your appointment, then Enter.
  <selfld>Weekdays:
    <choice checkvar=day match=M availvar=fullM >Monday
    <choice checkvar=day match=T availvar=fullT >Tuesday
    <choice checkvar=day match=W availvar=fullW >Wednesday
    <choice checkvar=day match=H availvar=fullH >Thursday
    <choice checkvar=day match=F availvar=fullF >Friday
  </selfld>
</panel>
```

By setting the variable, *fullW* to false, and all of the other AVAILVAR variables to true before the panel is displayed, you can indicate that Wednesday is no longer an available day for scheduling appointments.

The AVAILVAR attribute can be used in conjunction with the SETVAR and TOGVAR actions to dynamically de-emphasize or select choices as the user makes certain selections. For example, suppose that the person who does manicures does not work on Thursday. To visually indicate this to the user, we'll de-emphasize the **Manicure** choice when **Thursday** is selected, and de-emphasize **Thursday** when **Manicure** is chosen. By properly defining the tags, as shown here, the Dialog

Manager will automatically handle this at run time, with no application involvement.

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>

<varlist>
  <vardcl name=man varclass=sampcls>
  <vardcl name=noman varclass=sampcls>
</varlist>

<panel name=bothsel>Schedule Appointments
  <ab>

⋮
  </ab>
  <area>
    <selfld>Choose the day for the appointment:
      <choice>Monday
        <action setvar=man>
      <choice>Tuesday
        <action setvar=man>
      <choice>Wednesday
        <action setvar=man>
      <choice availvar=noman>Thursday
        <action setvar=man value=0>
      <choice>Friday
        <action setvar=man>
    </selfld>
    <selfld type=multi>Choose the services needed:
      <choice>Dry haircut
      <choice>Shampoo, haircut, and style
      <choice>Permanent or body wave
      <choice>Facial
      <choice availvar=man>Manicure
        <action togvar=noman value1=1 value2=0>
      <choice>Pedicure
    </selfld>
  </area>
</panel>
```

To define this behavior in the tags, we coded a SETVAR action under the **Thursday** choice. This action sets the variable, *man* (manicure available) to false whenever **Thursday** is chosen. This variable is also used as the AVAILVAR value for the **Manicure** choice. When **Thursday** is chosen the following actions occur:

- The SETVAR action sets *man* = 0.
- The variable *man* is used as AVAILVAR on **Manicure**, so **Manicure** is immediately de-emphasized.

The converse must also be defined, so we specified a TOGVAR action, under **Manicure**, that toggles the variable, *noman* (no manicure selected) between the values 0 and 1.

When the **Manicure** check box is selected:

- The variable *noman* = 1 so it is toggled to 0.

- The variable *noman* is used as AVAILVAR on **Thursday**, so **Thursday** is immediately de-emphasized.

When the **Manicure** check box is de-selected:

- The variable *noman*=0 so it is toggled to 1.
- Since *noman* is the AVAILVAR on **Thursday**, this makes **Thursday** an available choice again.

Selection Width

The SELWIDTH attribute of the SELFLD tag can be used to define how much space is taken up by each choice. The selection width determines the length of the selection cursor, so if you have choices with varying length text, you can use the SELWIDTH attribute to specify that the cursor is the same length for all of the choices. This attribute is also useful in aligning choices in multiple horizontal selection fields, as shown here:

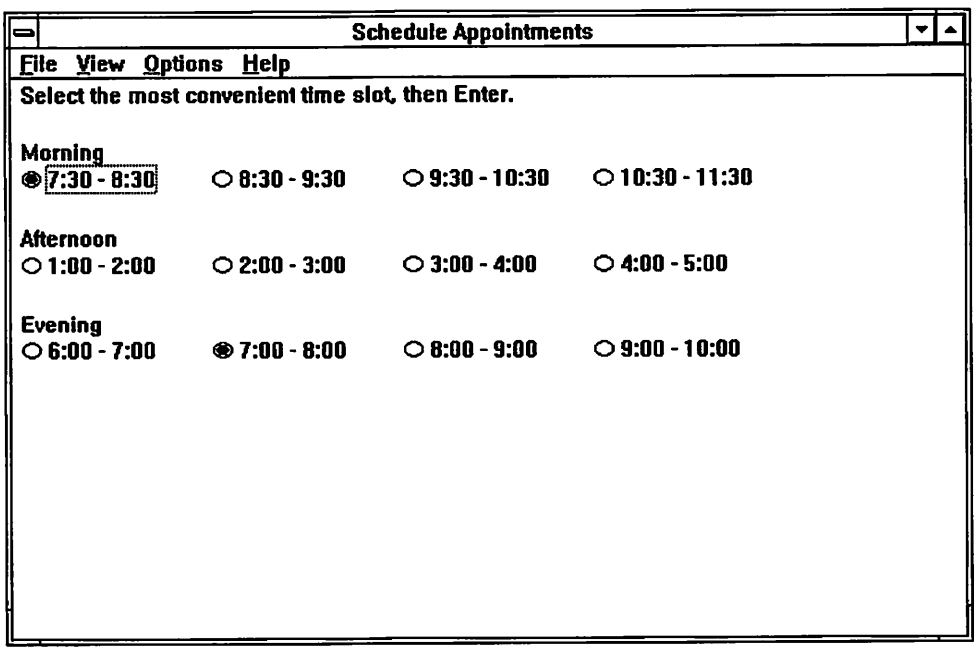


Figure 5-12. Selection Field SELWIDTH Attribute

Here is the markup used to define the panel:

```
<panel name=selwid>Schedule Appointments
  <ab>

:
  </ab>
  <area>
    <topinst>Select the most convenient time slot, then Enter.
    <selfld dir=horiz selwidth=12>Morning
      <choice>7:30 - 8:30
      <choice>8:30 - 9:30
      <choice>9:30 - 10:30
      <choice>10:30 - 11:30
    </selfld>
    <divider>
    <selfld dir=horiz selwidth=12>Afternoon
      <choice>1:00 - 2:00
      <choice>2:00 - 3:00
      <choice>3:00 - 4:00
      <choice>4:00 - 5:00
    </selfld>
    <divider>
    <selfld dir=horiz selwidth=12>Evening
      <choice>6:00 - 7:00
      <choice>7:00 - 8:00
      <choice>8:00 - 9:00
      <choice>9:00 - 10:00
    </selfld>
  </area>
</panel>
```

Data Columns

The DTACOL (data column) tag can be used to define default values for prompt width, entry width, selection width, and description width for the data fields and selection fields that are coded within the data column. If you have a group of data fields and selection fields that have the same width attributes, the DTACOL tag is a convenient short-cut.

The DTACOL tag has four attributes associated with it;

PMTWIDTH Applies to data fields and selection fields.

ENTWIDTH Applies to data fields only.

DESWIDTH Applies to data fields only.

SELWIDTH Applies to selection fields only.

These attributes serve the same purposes in DTACOL definitions as they do in DTAFLD and SELFLD definitions. The only difference is that when you use them with a DTACOL tag, they define those values for all of the data fields and selection fields coded between the DTACOL start and end tags.

The following markup uses a data column to define a default prompt width, entry width, and description width for the data fields and the selection field coded within the data column.

```
<!doctype dm system>

<varclass name=sampcls type ='char 30'>
<varclass name=statcls type ='char 2'>
<varclass name=zipcls  type ='char 5'>

<varlist>
  <vardcl name=name varclass=sampcls>
  <vardcl name=addr varclass=sampcls>
  <vardcl name=city varclass=sampcls>
  <vardcl name=stat varclass=statcls>
  <vardcl name=zipc varclass=zipcls>
</varlist>

<panel name=dcolxmp>Schedule Appointments
  <ab>

⋮
  </ab>
  <topinst>Enter your name and address and
  choose the most convenient day for your appointment:
  <area>
    <dtacol pmtwidth=8 entwidth=15 deswidth=10>
      <dtafld datavar=name>Name
        <dtafldd>Last, first, m.i.
      <dtafld datavar=addr>Address
        <dtafldd>If it applies, include apartment number
      <dtafld datavar=city>City
      <dtafld datavar=stat>State
        <dtafldd>Use 2 character abbreviation
      <dtafld datavar=zipc>Zip code
      <divider type=solid>
      <selfld pmtloc=before>Weekdays:
        <choice>Monday
        <choice>Tuesday
        <choice>Wednesday
        <choice>Thursday
        <choice>Friday
      </selfld>
    </dtacol>
  </area>
</panel>
```

Here is the result:

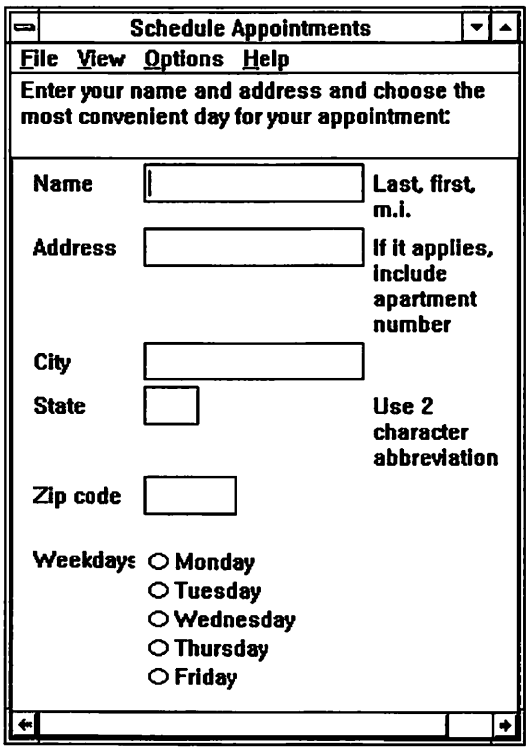


Figure 5-13. Data Column

You can override the values specified on the DTACOL tag by specifying the value on the enclosed DTAFLD or SELFLD tag. For example, in the preceding markup, if we specified PMTWIDTH=10 on the SELFLD tag, the selection field prompt width would be 10 character units, which overrides the *8* specified for the prompt width on the DTACOL tag.

Defining Selection Lists

A selection list is used to display a list of choices, where the list of choices comes from an array variable. A selection list is automatically scrollable in the horizontal and vertical directions, if necessary. Selection lists can be single-choice or multiple-choice.

The tags you use to define selection lists are:

SELLST To define the selection list. A matching end tag is required.

SELCOL To define the column of data displayed within the selection list. The SELCOL tag is coded within the SELLST tag. You can code only one SELCOL tag within a SELLST definition.

Figure 5-14 on page 5-27 contains a selection list..

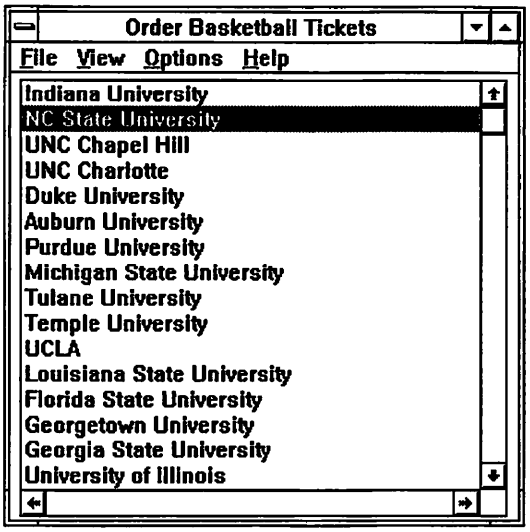


Figure 5-14. Selection List

Here's the markup used to create the selection list:

```
<!doctype dm system>

<varclass name=teamcls type='char 30'>
<varclass name=numclass type='numeric 5'>

<varlist>
  <vardcl name=teams   varclass=teamcls maxdim=200>
  <vardcl name=selitem varclass=numclass>
</varlist>


<panel name=sellst>Order Basketball Tickets
  <ab>

⋮
  </ab>
  <sellst selvar=selitem>
    <selcol datavar=teams>
  </sellst>
</panel>
```

In the preceding markup, the SELLST and SELCOL tags are used to define the selection list. The data that is displayed in the selection list is defined in the application within the variable, *teams*, which must belong to an array variable class.

You can allow the user to select only one item from the selection list, or select multiple items, by using the TYPE attribute on the SELLST tag. If you specify TYPE = SINGLE, the user can only select one item in the selection list. TYPE = MULTI allows the user to select more that one item. In the previous example, the TYPE attribute is not specified, so the default, SINGLE, is assumed.

The SELLST tag also has an optional SELVAR attribute. The SELVAR attribute is used for two purposes:

- The application uses the SELVAR to specify preselected items in the selection list.
- The Dialog Manager uses the SELVAR to communicate back to the application which items the user selected from the selection list.

If the selection list is single-choice, the selection variable must be scalar. To specify the preselected item in a single-choice selection list, set the selection variable to the index (1-based) of the item to be selected before the panel is displayed. By setting the variable to 0, you specify that no items are to be preselected. When control returns to the application, the selection variable contains the index of the item that the user selected. For example, if you set the selection variable to 5, the fifth item in the selection list will be selected when the panel is displayed. If the user selects item 10, then the selection variable is updated to contain 10.

For multiple-choice selection lists, the selection variable must be an array variable, and should have (at least) the same number of elements in the array as the number of elements in the selection list. You specify the preselected items by putting the indexes of the items to be preselected in the selection variable array. You must put a 0 in the entry following the index of the last preselected item, as 0 is used to end the list. When control returns to the application, the selection variable array contains the indexes of the items that were selected by the user. Again, a 0 is used to end the list, unless there are more selections than would allow for the ending 0 to be added without exceeding the number of elements in the SELVAR array. In this case, the index of the last item is in the selection variable's last element.

You can specify the number of items in the selection list using the NUMROWS attribute of the SELLST tag. The value for this attribute can be a constant or a variable. In the previous example, the NUMROWS attribute is not specified, so the Dialog Manager assumes the default, which is the number of elements in the data variable. This default is assumed for both single- and multiple-choice selection lists.

In the example on page 5-27, the default assumed for the number of items is 200, because the selection list is single-choice and the number of elements in the data variable array is 200 (defined using MAXDIM on VARDCL).

The following example specifies the number of items with the variable, *numteams*. It is a multiple-choice selection list, so the selection variable, *selitem*, has to be an array.

```
<!doctype dm system>

<varclass name=teamcls  type='char 30'>
<varclass name=numclass type='numeric 5'>
<varclass name=selcls   type='char 4'>

<varlist>
  <vardcl name=teams    varclass=teamcls maxdim=200>
  <vardcl name=selitems varclass=selcls  maxdim=200>
  <vardcl name=numteams varclass=numclass>
</varlist>

<panel name=sellst>Order Basketball Tickets
  <ab>

:
  </ab>
  <sellst selvar=selitems type=multi numrows='%numteams'>
    <selcol datavar=teams>
  </sellst>
</panel>
```

Here is the resulting panel from the above example:

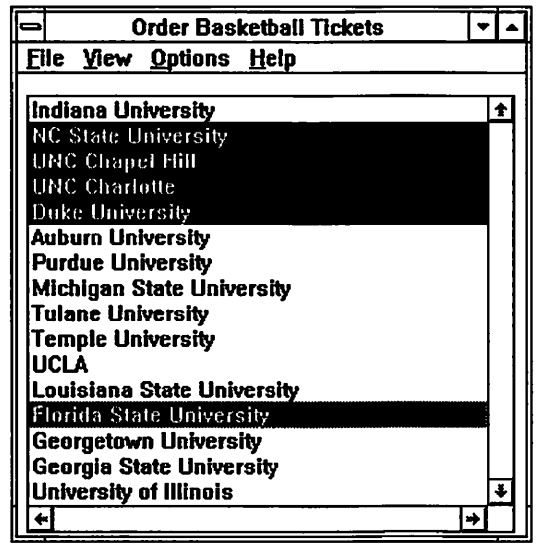


Figure 5-15. Selection List

Other Selection List Attributes

You can control the size of a selection list using the following SELLST attributes:

- MINROWS
- MAXROWS
- MINWIDTH
- MAXWIDTH.

"Sizing the Selection List and List Field" on page 5-36 provides a complete description of these attributes.

There are two other SELLST attributes you can use to define the behavior of a selection list:

TOPROW
This attribute allows you to specify which item is scrolled to the top of the selection list when the panel is displayed. You can specify TOPROW as a constant or a variable. If you do not specify the TOPROW attribute, the default (1) is used, and the first item is scrolled to the top when the panel is displayed. The item specified on TOPROW is only scrolled to the top if there are sufficient following items in the selection list to scroll that item to the top without leaving blank items at the bottom of the list.

VARCLASS
This attribute can be used to to override the variable class that is specified on the variable declaration (VARDCL) for the selection list's data variable (DATAVAR). See Chapter 4, "Variables and Variable Classes" on page 4-1 for a description of variables and variable classes.

Selection List Help

The Dialog Manager allows you to provide help on the selection list by using the HELP attribute on the SELLST tag. If you specify the name of the help panel for the selection list, the Dialog Manager knows which help panel to display when the user selects help on the selection list. If you do not specify help for the selection list, the extended help panel is displayed.

The following example adds the HELP attribute to the example we created earlier. When help is requested on the selection list, the help panel with the NAME value, *teamhelp*, is displayed.

```
  <sellst selvar=selitem help=teamhelp>
    <selcol datavar=teams>
  </sellst>
</panel>
```

Defining List Fields

A list field is used to display data in column format, as well as to allow the user to enter data in the column rows. The list field supports automatic scrolling, and displays horizontal and vertical scroll bars around the list field if all of the data in the list field is not currently visible.

The tags you use to define a list field are:

LSTFLD To define the list field. A matching end tag is required.

LSTCOL To define a column within a list field. You code a LSTCOL tag for each column of data in the list field.

A list field can contain one or more columns of data, where each column can be input-only, output-only, or input/output, as defined by the USAGE attribute on the LSTCOL tag. These are the values you can specify on the USAGE attribute:

IN Defines an input-only list column. An input-only column is blank when it is initially displayed, and the user can enter data into any of the rows in the input column.

OUT Defines an output-only list column. When the panel is initially displayed, output-only columns display the value of the variable associated with the list column. The user cannot interact with an output-only list column.

BOTH Defines an input/output list column. Input/output list columns display the value of the variable associated with the list column, when the panel is initially displayed, as well as allowing the user to enter data into any of the rows in the column. This is the default usage if you do not specify the USAGE attribute.

The data that is associated with each list column must come from an array variable, which is specified on the DATAVAR attribute of the LSTCOL tag. Like all variables used on the panel, the data variable must be declared using the VARDCL tag. All data variables must be an array, so the MAXDIM attribute must be used on the VARDCL tag to specify the maximum number of elements in the array.

You can define a column heading for any of the list columns in the list field by specifying the column heading text as the tag text on the LSTCOL tag. The column headings are centered above the list field column. When the list field is horizontally scrolled, the column headings scroll; however, they do not scroll when the list field is vertically scrolled.

This panel shows a list field with six columns. The first column is output-only, and the remaining columns are input/output.

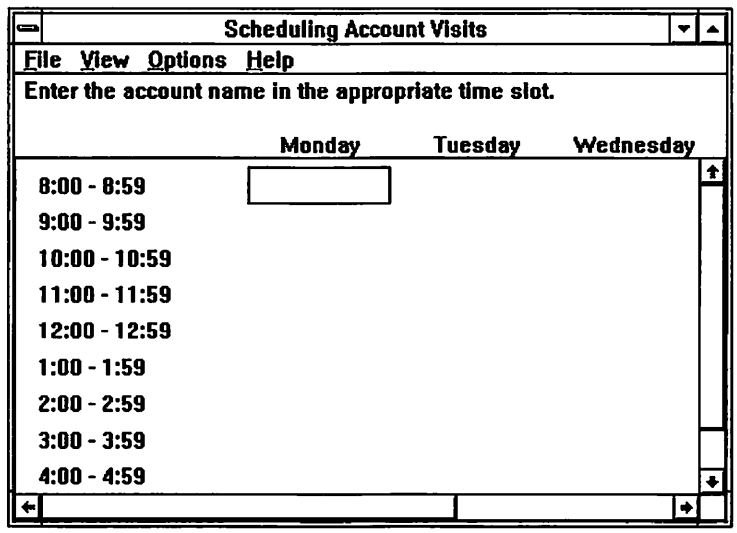


Figure 5-16. List Field

Here's the markup we used to create the panel:

```
<!doctype dm system>

<varclass name=timecls type=itime>
<varclass name=vc1     type ='char 20'>

<varlist>
  <vardcl name=timecol varclass=timecls maxdim=9>
  <vardcl name=moncol  varclass=vc1     maxdim=9>
  <vardcl name=tuecol  varclass=vc1     maxdim=9>
  <vardcl name=wedcol  varclass=vc1     maxdim=9>
  <vardcl name=thrcol  varclass=vc1     maxdim=9>
  <vardcl name=fricol  varclass=vc1     maxdim=9>
</varlist>

<panel name=lstfld1>Scheduling Account Visits
 <topinst>Enter the account name in the appropriate time slot.
 <lstfld>
   <lstcol datavar=timecol usage=out>
   <lstcol datavar=moncol>Monday
   <lstcol datavar=tuecol>Tuesday
   <lstcol datavar=wedcol>Wednesday
   <lstcol datavar=thrcol>Thursday
   <lstcol datavar=fricol>Friday
 </lstfld>
</panel>
```

List Column Width

As you can see from the preceding figure, the width of each column is defined in the same way data field entry widths are defined.

If COLWIDTH is specified in a LSTCOL tag, the column size serves as the upper bound to the data area size. The data is justified within the data area as specified in the ALIGN attribute of the LSTCOL tag. If the length of the data (as defined by the VARCLASS) exceeds the value of COLWIDTH, one of the following results will occur:

- For fields defined as INPUT or BOTH, part of the data may not be visible. In this case, the rest of the data is accessible through scrolling.
- For an OUTPUT field, the variable data is truncated at column width.

If COLWIDTH is not specified, the width of a column is determined as follows:

- For fields defined as INPUT or BOTH, the column width is set to be the minimum of the size determined by the defined VARCLASS or 25 (the default value). The field is scrollable up to the number of characters determined by the VARCLASS.
- For an OUTPUT field, the column width is the minimum of 25 or the length of the data.

Note: If the column heading text is longer than the width specified in a column width, the column is widened to avoid truncating the column heading text.

Rows in a List Field

You can use the NUMROWS attribute on the LSTFLD tag to define how many rows of data are contained in the list field. This attribute specifies the number of rows over which the list field scrolls. NUMROWS can be specified as a variable, so that it may be updated at run time by your application, as the number of rows of data changes.

If you do not specify the NUMROWS attribute, the Dialog Manager assumes that the number of rows is the smallest size of the column variable arrays defined for the list columns. In the previous example, we did not specify the NUMROWS attribute. Because the fewest number of elements defined for a column variable is 9, the Dialog Manager assumes there are 9 rows of data in the list field.

You can also use the TOPROW attribute of the LSTFLD tag to specify which row is scrolled to the top of the list field when the panel is displayed. You can specify TOPROW as a constant or a variable. If you do not specify the TOPROW attribute, the default, 1 is used and the first row is scrolled to the top of the list field when the panel is displayed.

List Field Help

The Dialog Manager allows you to provide help on each list column and the list field as a whole, by using the HELP attribute on the LSTFLD and LSTCOL tags. If you specify the name of the help panel for the list field or list column, the Dialog Manager knows which help panel to display when the user selects help on the list field or list column. If you do not specify help for a list column, the help for the list field is displayed when the user selects help. If there is no help defined for the list field, then the extended help panel is displayed.

Because output-only list columns cannot contain the cursor, the user cannot select help on them. You should ensure that the help information for an output-only list column is included in the help information for the list field or panel.

List Field Modification

The MODVAR attribute of the LSTFLD tag can be used to determine which rows were modified by the user. This attribute names an array variable which is updated by the Dialog Manager to contain the numbers of the rows that were modified by the user. For example, if rows 1, 5, and 8 were modified by the user, the MODVAR would contain 1 in the first element, 5 in the second element, 8 in the third element, and 0 in the fourth element (to end the list), when control returned back to the application. The dimension of the MODVAR determines the total number of items that the Dialog Manager can return as modified. For example, if the MODVAR is defined with a dimension of 50, the maximum number of rows that could be returned as modified is 50.

The MODVAR attribute can also be used in conjunction with the REQUIRED attribute (described in "Other List Column Attributes" on page 5-34) on a list column. If the REQUIRED=YES attribute is specified on a list column, input required is enforced only for the rows specified in the MODVAR. Continuing the previous example, if the panel was redisplayed, MODVAR still contained 1, 5, 8, and 0, and REQUIRED=YES was specified for the list column, input required would only be enforced for rows 1, 5, and 8 for that list column. This feature is useful if an application is validating data the user entered into an input-required list column and finds errors in certain rows. By setting the MODVAR to contain the numbers of

the rows in which there are errors, and redisplaying the panel, you can force the user to enter data into the rows where an error occurred.

List Field Grids

The RULES attribute can be used on the LSTFLD tag to specify that the list field has lines drawn within the list field. You specify one of these values on the RULES attribute:

NONE No lines are drawn in the list field. This is the default value.

HORIZ Horizontal lines are drawn between each row in the list field.

VERT Vertical lines are drawn between each column in the list field.

BOTH Horizontal lines are drawn between each row in the list field, and vertical lines are drawn between the columns in the list field.

Other List Column Attributes

There are several other attributes that can be used to define the behavior of the list field. Many of these attributes are the same as the attributes on the DTAFLD tag, since the list field allows data entry just like the data field does. The following list describes each of the remaining LSTCOL attributes and how they are used:

REQUIRED
This attribute indicates if this column is required to have input for any modified record. For input-required columns (REQUIRED=YES), the Dialog Manager will not validate the panel unless the user has entered data into the specified rows of that column. If you do not specify this attribute, input is not required on the list column. This attribute is only valid for input/both list columns. (See "List Field Modification" on page 5-33 for a description of the MODVAR attribute.)

MSG
This attribute identifies the message that should be displayed when the user does not enter any data into an input-required list column. If you do not specify this attribute, the Dialog Manager displays a default message. This attribute is valid only if REQUIRED=YES. Chapter 8, "Messages" on page 8-1 tells you how to define application messages.

ALIGN
This attribute allows you to align the variable data within the list column. The default value for ALIGN is *start*, which aligns the data from the left side of the column. You can also center the data within the column with the *center* value, or justify the data from the right side of the column with the *end* value. The attribute value END is useful for right justifying numbers within a column, since numbers are typically right-aligned.

AUTOCLR
This attribute allows you to specify the auto-clear attribute. The first character a user enters into a list field causes the remaining contents of the data in that row and column to be erased. The default value for AUTOCLR is *no*. This attribute is only valid for input/both list fields.

VARCLASS
This attribute allows you to override the variable class that is specified on the variable declaration (VARDCL) for the list column's data variable (DATAVAR). See Chapter 4, "Variables and Variable Classes" on page 4-1 for a description of variables and variable classes.

This example shows vertical rules in a list field, and center and right aligned list columns.

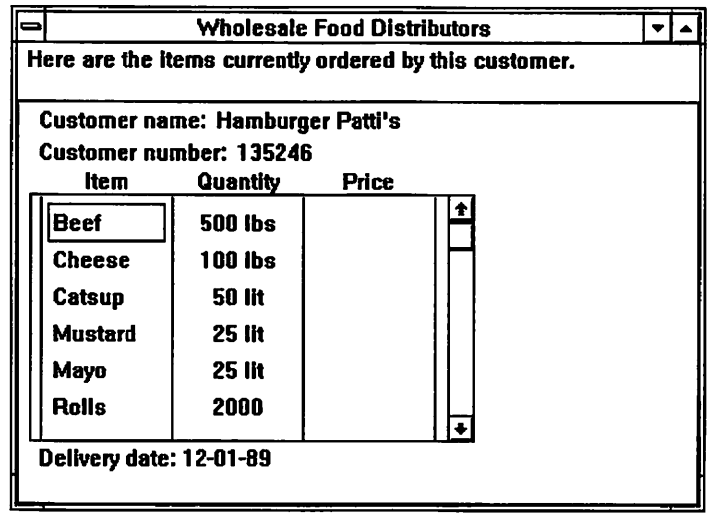


Figure 5-17. List Field

Here is the markup we used to create the panel:

```
<!doctype dm system>

<varclass name=vc1     type ='char 20'>
<varclass name=vc2     type ='char 6'>
<varclass name=qtycls  type ='char 3'>
<varclass name=priccls type ='numeric 3 2'>
<varclass name=daycls  type = idate>

<varlist>
  <vardcl name=cname    varclass=vc1>
  <vardcl name=cnum     varclass=vc2>
  <vardcl name=itemcol  varclass=vc2     maxdim=500>
  <vardcl name=qtycol   varclass=qtycls  maxdim=500>
  <vardcl name=pricecol varclass=priccls maxdim=500>
  <vardcl name=zidate   varclass=daycls>
</varlist>

<panel name=lstfld2>Wholesale Food Distributors
  <ab>

:
  </ab>
 <topinst>Here are the items currently ordered by this customer.
 <area>
 <dtafld datavar=cname usage=out>Customer name:
 <dtafld datavar=cnum  usage=out>Customer number:
 <lstfld rules=vert>
   <lstcol datavar=itemcol>Item Number
   <lstcol datavar=qtycol align=center>Quantity
   <lstcol datavar=pricecol align=end>Price
 </lstfld>
 <dtafld datavar=zidate usage=out>Delivery date:
 </area>
</panel>
```

The list field size can be controlled through attributes explained in "Sizing the Selection List and List Field."

Sizing the Selection List and List Field

Some of the fields defined using the compiler, such as data field and selection field are static in size. When the panel is sized, these fields remain the same size, and do not grow or shrink. The list field and selection list fields are sizeable, however, and they grow and shrink as the panel changes size.

There are four attributes you can specify on the SELLST and LSTFLD tags that allow you to control the sizing of the selection list and list field:

- MINROWS
- MAXROWS
- MINWIDTH
- MAXWIDTH.

If you do not specify any of these attributes, the selection list and list field are sized according to the default behavior. The default behavior allows these fields to continue to be sized larger vertically and horizontally until all of the data within the field is visible (up to a maximum of 512 characters). If the value of MINROWS or

MAXROWS is greater than the value of NUMROWS, the list field is sized to display all of the rows without additional space. This evaluation is made at run time because the NUMROWS value can be a variable.

Using the four sizing attributes mentioned above, you can set limits on the sizing behavior of these fields.

MINROWS When the panel is sized smaller vertically, the selection list and list field display fewer rows of data until the number of rows specified on MINROWS is displayed.

MINWIDTH As the panel is sized smaller horizontally, the selection list and list field become narrower until the value specified on MINWIDTH is reached.

MAXROWS When the panel is sized larger vertically, the list field displays more rows of data until all of the data is displayed, or the number of rows specified on MAXROWS is displayed.

When a panel containing a selection list is sized larger vertically, the selection list displays more data until the number of rows specified on MAXROWS is displayed.

MAXWIDTH When the panel is sized larger horizontally, the list field displays more data until all of the data is displayed, or the amount of data specified on MAXWIDTH is displayed.

When a panel containing a selection list is sized larger horizontally, the selection list displays more data until the amount of data specified on MAXWIDTH is displayed.

Chapter 6. Information Regions

Much of the information displayed on panels is *static*, or fixed text that the user has no direct interaction with. This includes text such as top instructions and bottom instructions, prompt text, and data field description text. DTL provides you with another method of defining static text for application panels using *information regions*.

Defining an information region on a panel allows you more flexibility for defining static text on a panel. The tags you use to define the text of information regions are much more versatile than the tags you use to define other types of static text, which means you can be more creative in the text you define.

Use the INFO tag and its required end tag to define an information region on a panel. If you want the information region to be scrollable, code the information region within an AREA definition. You can also code an information region within a REGION definition.

The INFO tag has an optional WIDTH attribute that defines the width (in character units) of the information region. The default width is 45 characters.[1]

Panel Design Note

If the value you assign the INFO WIDTH attribute is greater than the WIDTH value of the panel the information region is coded in, the text of the information region will exceed the initial window width. If you code the information region within an AREA definition a horizontal scroll bar appears on the panel indicating that more text is present outside of the displayed panel.

The INFO tag only defines an information region. It does not define the text of the information region. DTL provides you with a set of tags that define the text in information regions. These tags include:

- CAUTION
- DL (definition list)
- FIG (figure)
- FIGCAP (figure caption)
- Hn (heading)
- LINES
- NOTE
- NT (note)
- OL (ordered list)
- P (paragraph)
- PARML (parameter list)
- SL (simple list)
- UL (unordered list)
- WARNING
- XMP (example).

[1] The compiler ignores the WIDTH attribute when you code INFO within help panels. For more information on help panels, refer to Chapter 7, "Help Panels" on page 7-1.

These tags can *only* be coded within an INFO definition. In the next section, we tell you how to use each of these tags within information regions.

At the end of this chapter, we show you how you can use information regions to complement other panel elements on application panels.

Defining Basic Text

Paragraphs

The tag you use most often in information regions is the P (paragraph) tag. Use the P tag to arrange text as you would arrange a paragraph in your usual writing (to join one or more sentences related by their subject matter into a single block of text).

When the paragraph text is formatted for display, the text starts at the current margin and the words automatically wrap to fit within the margin.

The P tag has no attributes, and requires no matching end tag.

We'll illustrate the use of the P tag by defining one within an information region on a panel.

```
<panel name=infopan width=42 depth=16>Information
  <area>
    <info width=38>
      <p>This is a paragraph.
    </info>
  </area>
</panel>
```

Here is how the paragraph formats on the panel:

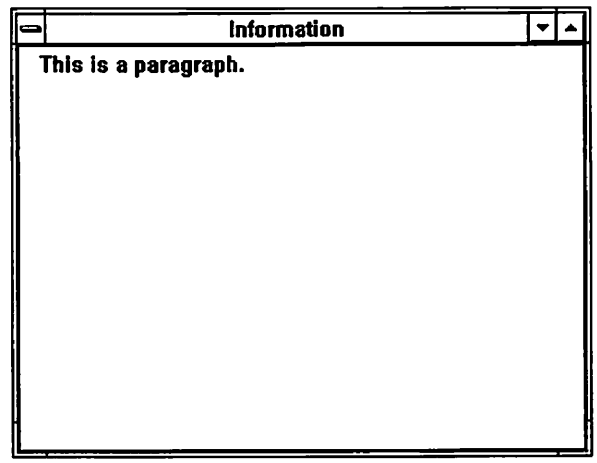


Figure 6-1. Paragraph

We'll add another sentence to the paragraph.

```
<panel name=infopan width=42 depth=16>Information
  <area>
    <info width=38>
      <p>This is a paragraph.
      This sentence is also part of the paragraph.
    </info>
  </area>
</panel>
```

Notice that we coded the new sentence on a different line in our markup. It doesn't matter, because the compiler treats it as part of the same paragraph, and formats it accordingly.

The compiler inserts a single space after each sentence in a paragraph. Here is the formatted result:

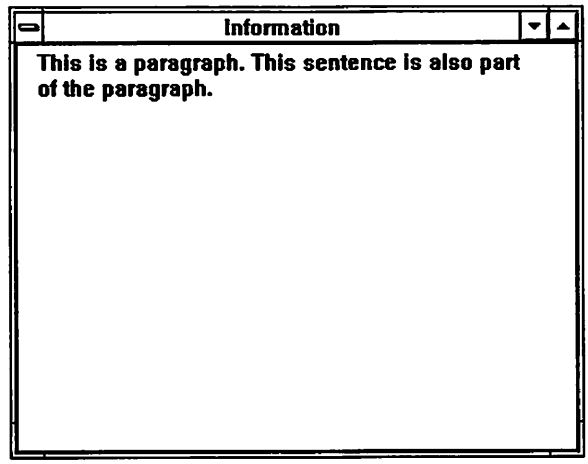


Figure 6-2. Paragraph

As you can see, the text of the paragraph is left-justified on the panel and the words automatically wrap to fit within the defined dimensions of the information region.

We'll add two more paragraphs to the information region.

```
<panel name=infopan width=42 depth=16>Information
  <area>
    <info width=38>
      <p>This is a paragraph.
      This sentence is also part of the paragraph.
      <p>Here is another paragraph.
      Paragraphs are very useful for providing
      information on panels.
      <p>Yet another paragraph.
      You can code as many paragraphs as you
      like within an information region.
    </info>
  </area>
</panel>
```

Figure 6-3 on page 6-4 shows the result:

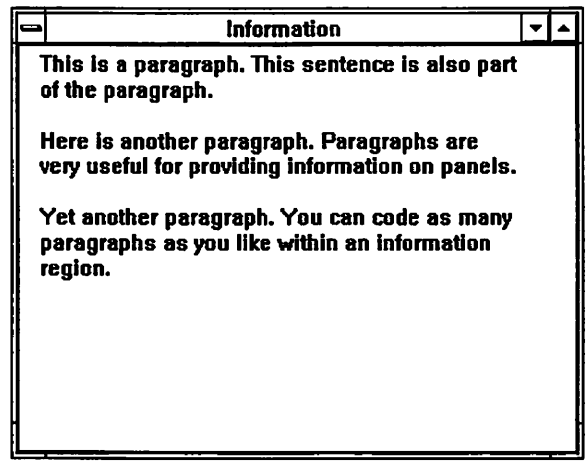


Figure 6-3. Multiple Paragraphs

In addition to the placement and wrapping of the text, the compiler separates the paragraphs with a blank line.

Headings

The Hn (heading) tag allows you to place headings in an information region. You use these headings to define topics and subtopics of information. You can define four levels of headings:

H1 The text is centered in the information region. Use this heading level to identify a main topic of information.

H2, H3, H4
The text formats against the left margin of the information region. Use one of these heading levels to identify subtopics of information.

Additionally, headings must be coded sequentially. That is, an H2 tag must be preceded by an H1 tag, an H3 tag must be preceded by an H2 tag, and an H4 tag must be preceded by an H3 tag.

The DTL compiler adds a blank line to the information region before formatting the text for any of the heading levels. There are no attributes associated with any of the heading tags, and none of them require an end tag.

The following markup contains an information region using all four heading levels, and paragraphs following each one.

```
<panel name=infopan width=42 depth=20>Information
  <area>
    <info width=38>
      <h1>A Main Topic
      <p>Notice how the heading is in the
      center of the information region?
        <h2>A Subtopic
        <p>This heading is left-justified.
          <h3>Another Subtopic
          <p>This heading is also left-justified.
            <h4>Yet Another Subtopic
            <p>Another left-justified heading.
        <h2>One More Subtopic
        <p>Here's another level two heading.
    </info>
  </area>
</panel>
```

Here's the formatted result:

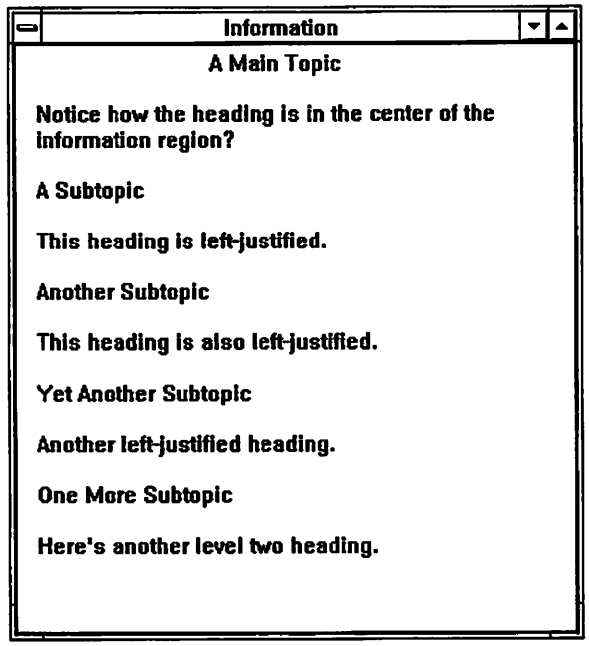


Figure 6-4. Headings (H1-H4)

Lines

Occasionally, you'll want to present text to users that you don't want formatted by the compiler, or that you want to show "as is." You can use the LINES (lines) tag and its required end tag to do this. All text coded within a LINES definition is treated as unformatted text, and you can position the text however you like on each line.

However, because a LINES is formatted using proportional fonts, the alignment of each line of text can vary. If you want the text to display *exactly* as you coded it, use the XMP (example) tag, which formats the text in a monospace font. The XMP tag is described on page 6-6.

There are many ways to use a LINES definition. Here we use it for a quotation:

```
<panel name=specact width=42 depth=16>Special Activities
  <area>
    <info width=38>
      <lines>
      Between the dark and daylight,
      When the night is beginning to lower,
      Comes a pause in the days' occupations,
      That's known as the children's hour.
                              -Longfellow
      </lines>
      <p>Every Tuesday evening at seven
      o'clock, we present the Children's Hour,
      a one hour recital of selected children's
      stories in our children's section.
    </info>
  </area>
</panel>
```

And our quotation appears just the way we marked it up:

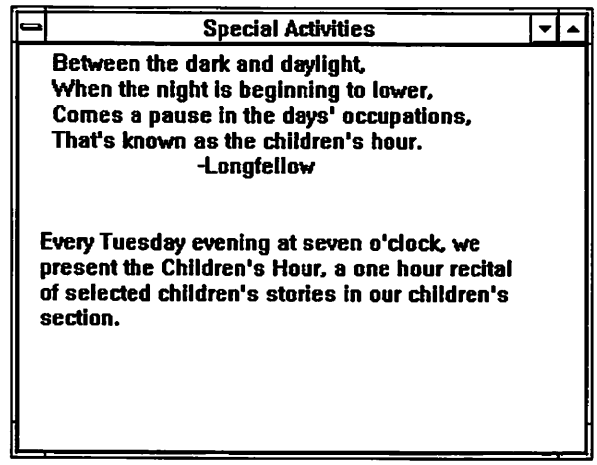


Figure 6-5. LINES

The LINES tag has no attributes and requires an end tag.

Note: When coding a LINES definition, be sure that you do not exceed the defined INFO width, otherwise the text will be truncated.

Examples

The XMP (example) tag is similar to the LINES tag, in that it allows you to code unformatted text. However, there are some differences between the XMP tag and LINES tag:

- The text of an XMP definition is indented two spaces from the current margin, as opposed to the text of a LINES definition, which is not indented from the current margin.
- The compiler formats the text of an XMP definition in a monospace font, as opposed to the proportional font used for LINES definitions.

Like a LINES definition, you should be careful of coding lines of text in an XMP definition that exceed the defined width of the information region.

The XMP tag has no attributes, and it requires a matching end tag.

The monospace formatting of examples is useful for showing examples of computer input and output. Here's our example of an example:

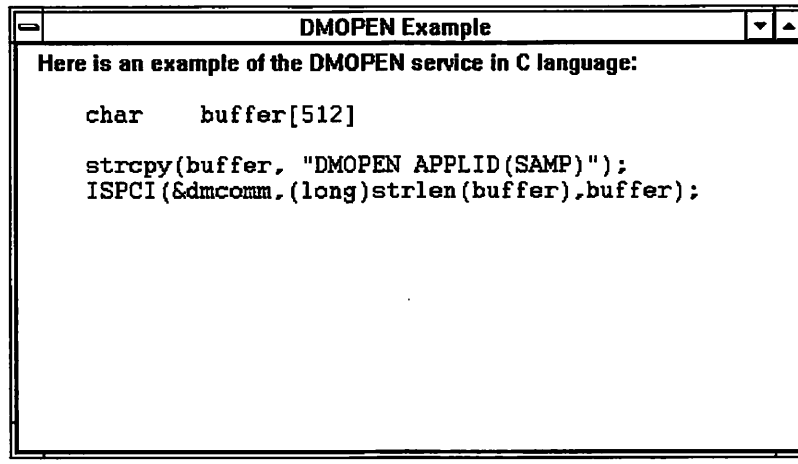


Figure 6-6. XMP

We used an example of the Dialog Manager DMOPEN service for this example.[2] The markup for this panel looks like this:

```
<panel name=dmopenx width=60 depth=16>DMOPEN Example
  <area>
    <info width=56>
      <p>Here is an example of the DMOPEN service
      in C language:
      <xmp>
      char    buffer[512]

      strcpy(buffer, "DMOPEN APPLID(SAMP)");
      ISPCI(&amp;dmcomm,(long)strlen(buffer),buffer);
      </xmp>
    </info>
  </area>
</panel>
```

Figures

The FIG (figure) tag is yet another way you can code text that isn't formatted. It works just like the LINES tag, except you can add a ruled border above and below the figure to separate it from the rest of the panel. You can also provide a caption for the figure using the FIGCAP tag.

Like the LINES and XMP tags, the FIG tag requires an end tag.

[2] In our example of an example, we needed to show an ampersand (&) as part of the example code. Because the DTL compiler interprets ampersands as entities, we used the predefined symbol for the ampersand in our markup. "Predefined Entities" on page 2-12 contains a list of the predefined symbols DTL provides.

To define the ruled borders for the figure, use the FRAME attribute of the FIG start tag.[3] The FRAME attribute has two values, RULE, which is the default, and NONE. Because RULE is the default value, you don't need to specify this attribute if you want ruled lines above and below the figure. To create a figure without rules, specify NONE as the FRAME value.

The figure in this panel will format with a ruled border:

```
<panel name=figurex width=50 depth=15>Books
  <area>
    <info width=46>
      <p>One philosopher sums up the experience
      of reading this way:
      <fig>
      Some books are to be tasted,
      others to be swallowed,
      and some few to be chewed and digested.
      </fig>
    </info>
  </area>
</panel>
```

Here's the formatted panel:

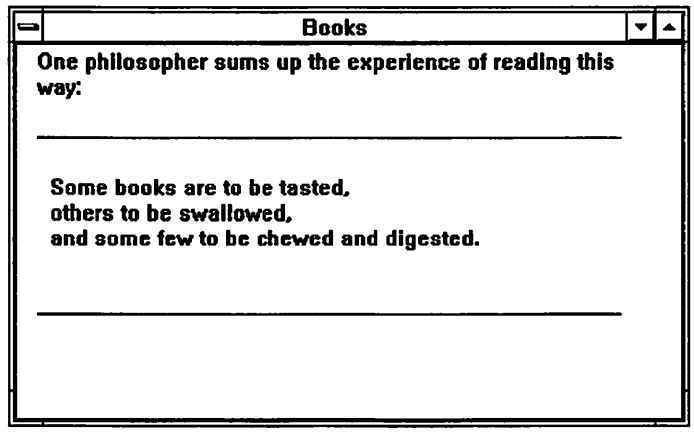


Figure 6-7. Figure with Rules

[3] The compiler ignores the FRAME and WIDTH attributes of the FIG tag when it is coded within a help panel. For more information on help panels, refer to Chapter 7, "Help Panels" on page 7-1.

We'll show you how the same figure looks without a ruled border by specifying NONE for the FRAME attribute.

```
<panel name=figurex width=50 depth=15>Books
  <area>
    <info width=46>
      <p>One philosopher sums up the experience
      of reading this way:
      <fig frame=none>
      Some books are to be tasted,
      others to be swallowed,
      and some few to be chewed and digested.
      </fig>
    </info>
  </area>
</panel>
```

Here's how the figure looks now:

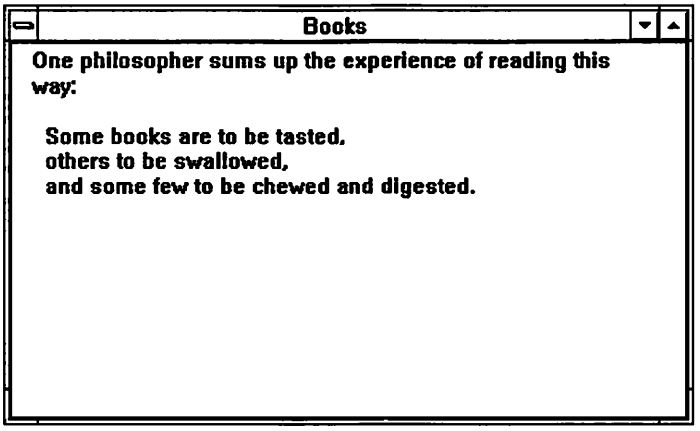


Figure 6-8. Figure without Rules

The FIG tag also has an optional WIDTH attribute that allows you to specify how the figure is aligned in the information region. The valid values for WIDTH are PAGE and COL. PAGE, which is the default value, aligns the figure along the left margin of the information region. COL indicates that the figure is aligned along the current left margin; that is, the current margin defined by the tag the figure is nested in. This is useful for aligning figures within list items, for example.

The Figure Caption (FIGCAP) Tag

We can add a caption to the figure in Figure 6-7 on page 6-8 by nesting a FIGCAP tag and caption text within the figure definition, like this:

```
<panel name=figurex width=50 depth=15>Books
  <area>
    <info width=46>
      <p>One philosopher sums up the experience
      of reading this way:
      <fig>
      Some books are to be tasted,
      others to be swallowed,
      and some few to be chewed and digested.
      <figcap>Francis Bacon
      </fig>
    </info>
  </area>
</panel>
```

We used this FIGCAP definition to give credit where it is due. The figure caption appears just below the bottom figure rule:

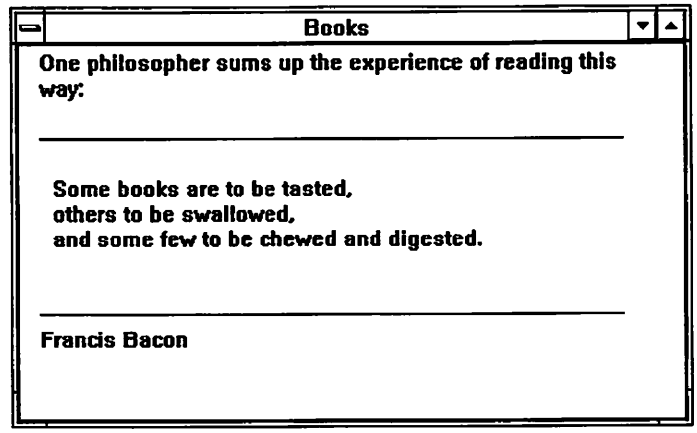


Figure 6-9. Figure Caption

Defining Lists

Sometimes you want to present information to the user that is not appropriate in paragraph form; such as listing more than two items, presenting a sequence of items or actions, or defining terms. For these situations (and many others), you can use the DTL list tags to format your text appropriately.

The following list describes the types of lists you can create:

Simple lists — Format as indented lists of items without any preceding item identifiers.

Unordered lists — Format as indented lists of items with each item preceded by a bullet (o), a hyphen (-), or a dash (—), depending on the level of nesting.

Ordered lists — Format as indented lists of items with each item preceded by a number or letter indicating its sequence in the list.

Definition lists — Format in two columns, with terms in one column and their matching descriptions in the other. You can also specify headings for each column in the list. (This list you are reading is a definition list.)

Parameter lists — Format similar to definition lists, but designed specifically to identify and define parameter terms.

The list items in simple lists, unordered lists, and ordered lists are created with the list item (LI) tag. The list item tag does not require an end tag, it is implicitly ended by another LI tag, an LP tag, or the end tag of the list it is coded within.

The characters that precede list items in unordered lists and ordered lists are language-sensitive, and depend on the /LANGUAGE option of the DTLC command when the source file is compiled. You can also use the /LANGUAGE option to change the appearance of these characters, for example, to change the case of the tag characters. See the description of the /LANGUAGE option in "DTLC" on page 13-2 for more information.

Simple Lists

Simple lists are the least complex types of lists. Use a simple list when the information you are presenting does not follow a sequential pattern or when bullets are not required to discriminate one list item from another.

Figure 6-10 on page 6-12 illustrates a simple list.

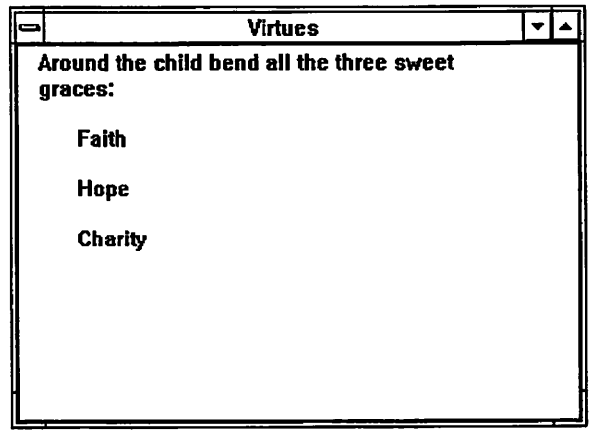


Figure 6-10. Simple List

This is the markup for the panel:

```
<panel name=slistx width=42 depth=15>Virtues
  <area>
    <info width=38>
      <p>Around the child bend all the
      three sweet graces:
      <sl>
        <li>Faith
        <li>Hope
        <li>Charity
      </sl>
    </info>
  </area>
</panel>
```

We defined each of the list items by nesting the LI tags within the simple list definition.

As you can see, our simple list formatted with a blank line between each of the list items. For cases where you need to conserve space, you can use the COMPACT attribute to format the list without blank lines between the list items.

Code the COMPACT attribute within the SL start tag (before the tag close delimiter), like this:

```
<panel name=slistx width=42 depth=15>Virtues
  <area>
    <info width=38>
      <p>Around the child bend all the
      three sweet graces:
      <sl compact>
        <li>Faith
        <li>Hope
        <li>Charity
      </sl>
    </info>
  </area>
</panel>
```

Now the simple list is compacted:

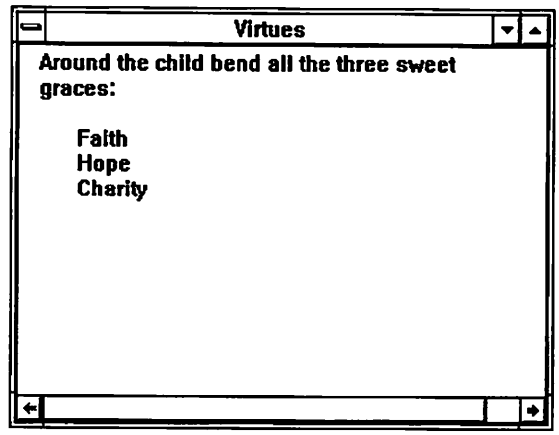


Figure 6-11. Compact Simple List

You can also nest simple lists within other simple lists. The list items will format at different indentation levels, based on the level of nesting.

Unordered Lists

Unordered lists are similar to simple lists, except each list item is preceded by a bullet symbol (o). You don't have to supply the symbol – DTL does that for you.

Use an unordered list if the list items are long and you don't want to imply any particular sequence in the list.

Here's an unordered list:

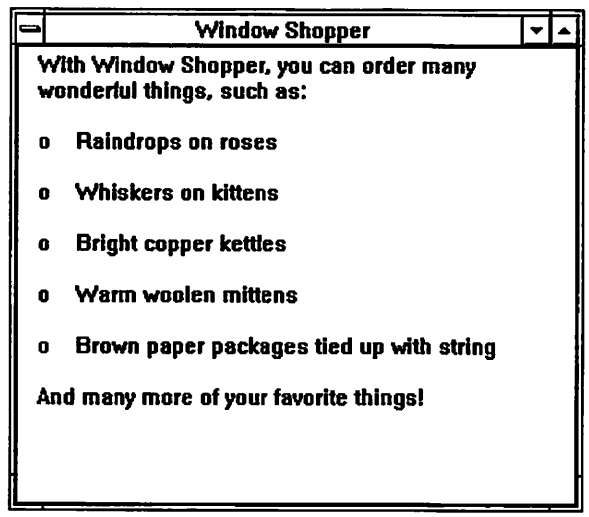


Figure 6-12. Unordered List

And here's the markup for this unordered list:

```
<panel name=winshop width=42 depth=18>Window Shopper
  <area>
    <info width=38>
      <p>With Window Shopper, you can order many wonderful things,
      such as:
      <ul>
        <li>Raindrops on roses
        <li>Whiskers on kittens
        <li>Bright copper kettles
        <li>Warm woolen mittens
        <li>Brown paper packages tied up with string
      </ul>
      <p>And many more of your favorite things!
    </info>
  </area>
</panel>
```

We could also have made this list compact like our simple list example, since the COMPACT attribute value is also valid on the UL tag.

You can also define levels of unordered lists; that is, you can nest unordered lists within other unordered lists. When you do this, the symbols preceding the list items in each level of the list vary, depending on the level of nesting. Specifically, the list items in the first (or only) level of unordered list are preceded by bullets (o), as shown in Figure 6-12 on page 6-13. If you nest another unordered list within an unordered list the list items in that list are preceded by hyphen symbols (-). A third level unordered list has dashes (—) preceding the list items. And, like any manner of nesting, the nested tags are aligned according to the level they are nested within.

To show how this works, we'll create a panel with three levels of unordered lists. The second and third levels in this example are compact lists.

```
<panel name=ulists width=42 depth=18>Nested Unordered Lists
  <area>
    <info width=38>
      <ul>
        <li>First level, first item
        <li>First level, second item
         <ul compact>
           <li>Second level, first item
           <li>Second level, second item
             <ul compact>
               <li>Third level, only item
             </ul>
         </ul>
        <li>Back to the first level
      </ul>
    </info>
  </area>
</panel>
```

Here's how this panel looks:

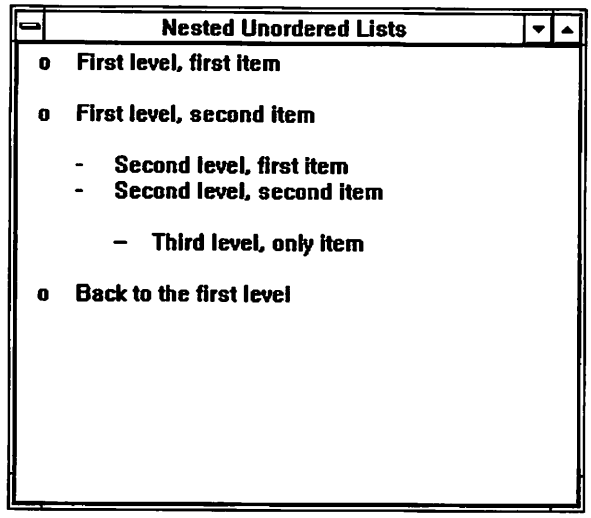


Figure 6-13. Nested Unordered Lists

If you nest more than three levels of unordered lists, the sequence of bullets, hyphens, and dashes repeats. For example, a fourth level would be preceded by bullets, a fifth level by hyphens, and so on.

As with any form of nesting, be sure to end each tag definition properly, or unexpected results can occur.

Ordered Lists

Ordered lists imply an outline sequence to the list items by preceding each of the list items with a number or character depending on the level of nesting.

Here's an ordered list:

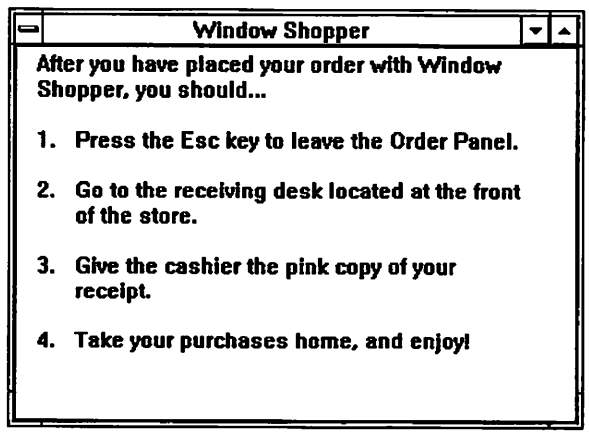


Figure 6-14. Ordered List

You don't have to supply the numbers for the list items in your markup, they are generated automatically. This saves you time when you revise ordered lists, because you can insert, delete, or rearrange list items without renumbering them yourself.

Here's the markup we used for this list:

```
<panel name=winshop2 width=42 depth=15>Window Shopper
  <area>
    <info width=38>
      <p>After you have placed your order with Window Shopper, you should...
      <ol>
        <li>Press the Esc key to leave the Order Panel.
        <li>Go to the receiving desk located at the front of the store.
        <li>Give the cashier the pink copy of your receipt.
        <li>Take your purchases home, and enjoy!
      </ol>
    </info>
  </area>
</panel>
```

Like other types of lists, you can nest ordered lists within other ordered lists. And, like unordered lists, the levels of the lists you nest determine the characters that precede the list items.

Specifically, the DTL uses the following sequence when preceding list items in nested ordered lists:

- First (or only) level list items are preceded by sequential numbers followed by periods.
- Second level list items are preceded by sequential lowercase alphabetic characters followed by periods.
- Third level list items are preceded by sequential numbers followed by close parentheses symbols.
- Fourth level list items are preceded by sequential lowercase alphabetic characters followed by close parentheses symbols.

The sequence of nesting is repeated for levels of nesting beyond the fourth level. For example, the list items in a fifth level of nesting are preceded by sequential numbers followed by periods.

To show you what this looks like, we'll nest four levels of ordered lists in this markup:

```
<panel name=olists width=42 depth=18>Nested Ordered Lists
  <area>
    <info width=38>
      <ol>
        <li>Step one (first level)
        <li>Step two (first level)
          <ol compact>
            <li>Step one (second level)
            <li>Step two (second level)
              <ol compact>
                <li>Step one (third level)
                  <ol compact>
                    <li>Step one (fourth level)
                    <li>Step two (fourth level)
                  </ol>
                <li>Step two (third level)
              </ol>
            <li>Step three (second level)
          </ol>
        <li>Step three (first level)
      </ol>
    </info>
  </area>
</panel>
```

In the interest of conserving space, we made several of the lists compact. Here's how the DTL compiler formats this panel:

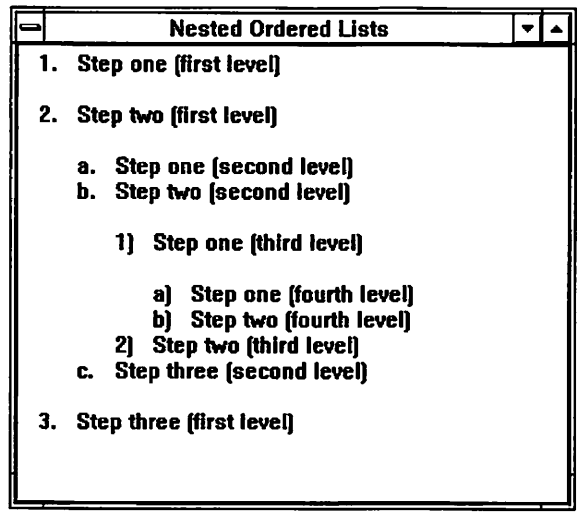


Figure 6-15. Nested Ordered Lists

Definition Lists

Definition lists allow you to identify a list of words or phrases and their corresponding definitions. A definition list formats as a two-column list: the terms you define appear in the left column, and the definitions for the terms appear in the right column. Definition lists are slightly more complex than the previous lists we've discussed, because of the additional tags required to construct them.

The following list tells you which tags to use to create a definition list:

DL Begins a definition list. The required end tag ends the list.

DT Identifies the term being defined. The definition term is formatted in the left column of the list. It does not require an end tag.

DD Identifies the term description. Each definition description is formatted in the right column of the list, immediately opposite or below its associated term. It does not require an end tag.

You can also create headings for definition list columns. There are two additional tags that you can use to do this. They are:

DTHD Defines a header for the definition term column.

DDHD Defines a header for the definition description column.

Both of these tags are optional for creating definition lists. We'll show you how you can use them to enhance definition lists later on in this section.

Here's an example of a definition list:

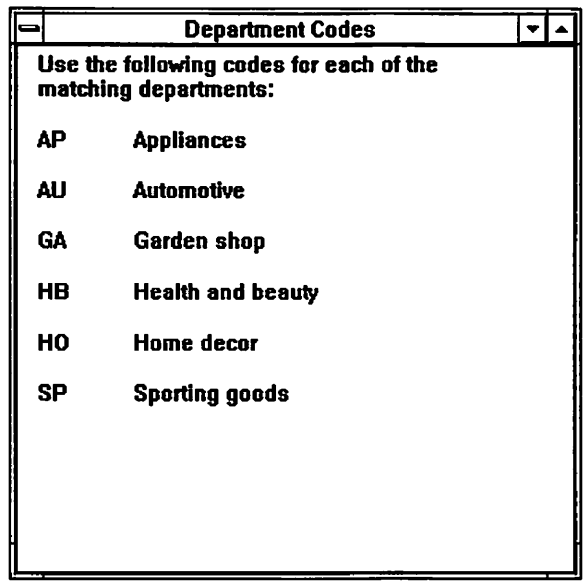


Figure 6-16. Definition List

Here is the markup:

```
<panel name=deptcode width=42 depth=24>Department Codes
  <area>
    <info width=38>
      <p>Use the following codes for each of the
      matching departments:
      <dl>
        <dt>AP
        <dd>Appliances
        <dt>AU
        <dd>Automotive
        <dt>GA
        <dd>Garden shop
        <dt>HB
        <dd>Health and beauty
        <dt>HO
        <dd>Home decor
        <dt>SP
        <dd>Sporting goods
      </dl>
    </info>
  </area>
</panel>
```

The DL tag has three optional attributes: TSIZE, which specifies the space allocated for the term column; BREAK, which indicates if the definition formats on the same line as its associated term, and COMPACT, which determines if there is a space between each set of terms and descriptions.

Use the TSIZE attribute to specify how much space you want for the definition term column. The default value is 10 character units. If you want to specify more (or less) space than the default, use the TSIZE attribute to assign the value you want.

Use the BREAK attribute to specify where the definition descriptions are supposed to start (on the same line as the definition terms or on the next line).

The BREAK attribute can be specified as NONE, ALL, or FIT. The default value is NONE, which means that the definition descriptions start on the same lines as the definition terms. ALL means that all of the definition descriptions start on the line after the definition terms. FIT means that the definition descriptions are to start on the next line only when the definition term does not fit in the allocated space and spills over into the description area.

The definition list in Figure 6-16 on page 6-19 uses the default BREAK=NONE. Here's the same list formatted with the ALL value:

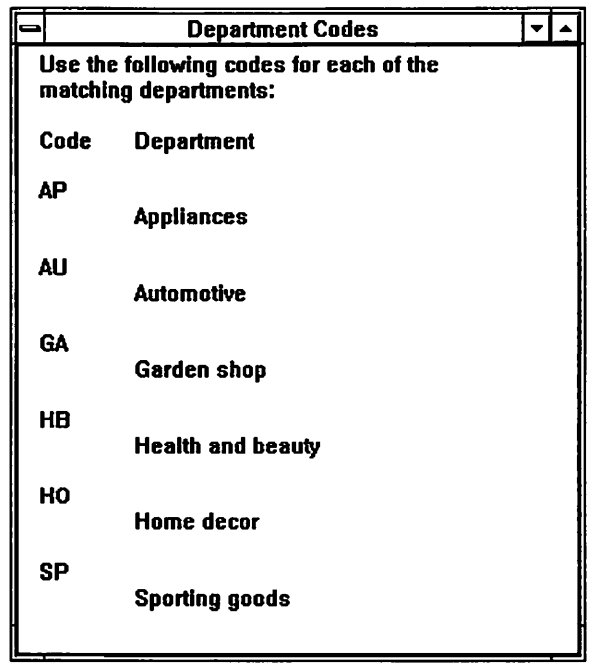


Figure 6-17. Definition List (BREAK=ALL)

And here, we use the FIT value for the BREAK attribute on a different definition list. We also decreased the TSIZE to 8 to illustrate the formatting. Here's the markup:

```
<panel name=reverb width=42 depth=20>Reverberations
  <area>
    <info width=38>
      <p>Reverberations is one of the most popular brands of electronic
      components available today.
      We stock the following Reverberations components:
      <dl tsize=8 break=fit>
        <dthd>Component
        <ddhd>Features
        <dt>CD Player
        <dd>With auto-search, auto-off, power door, and
        a two-year warranty.
        <dt>Receiver
        <dd>Digital, 6 speaker hookup, and built-in equalizer.
        <dt>Tape deck
        <dd>Supports metal and chrome cassettes, and comes with
        a two-year warranty.
      </dl>
    </info>
  </area>
</panel>
```

We also added the DTHD and DDHD tags we told you about earlier to provide headings for the definition list. Here is the formatted panel:

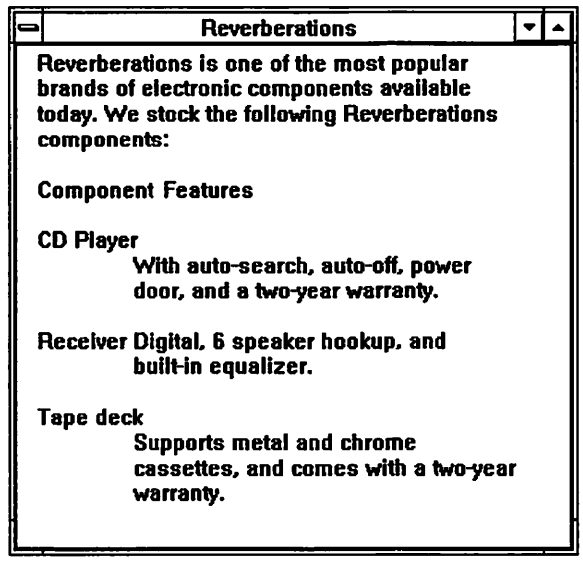


Figure 6-18. Definition List (BREAK=FIT)

Parameter Lists

Parameter lists are another way of defining terms in a list form. You use a parameter list when the terms you are defining are related to the application in some way (for example, showing code or parameters). This makes it easy to search for and make corrections to these terms if you ever update the application.

The tags you use to create parameter lists are the PARML tag and its required end tag, the PT (parameter term) tag, and the PD (parameter description) tag. The parameter list tags work just like the definition list tags in defining terms and descriptions.

The PARML tag also contains the TSIZE and BREAK attributes. The TSIZE default value is 10 character units, as it is for definition lists. However, the BREAK default value for parameter lists is ALL, instead of NONE, as in definition lists. Thus, the parameter descriptions format on the lines following the parameter terms unless you specify otherwise.

Here's the markup for a typical parameter list:

```
<panel name=ordnum width=42 depth=20>Order Numbers
  <area>
    <info width=38>
      <p>The order number assigned to each inventory item
      represents specific information about the item.
      <p>Specifically,
      <parml>
        <pt>123
        <pd>The first three digits represent the
        department the item is stocked in.
        <pt>456
        <pd>The fourth, fifth, and sixth digits
        represent the item.
        <pt>78
        <pd>The seventh and eighth digits represent
        the lot number of the item.
      </parml>
      <p>Please refer to the order number when
      inquiring about the item.
    </info>
  </area>
</panel>
```

And here's the formatted parameter list:

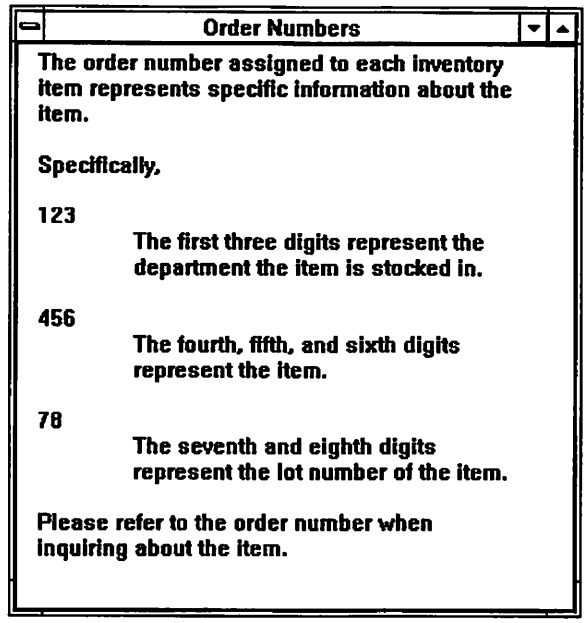


Figure 6-19. Parameter List

Nesting Tags within Lists

The format of your lists isn't confined to only list items. You can also nest other tags within the list items. For example, if a list item requires an additional paragraph, you can nest a P tag following the list item.

This markup contains an ordered list with a paragraph nested within the second list item.

```
<panel name=winshop2 width=42 depth=18>Window Shopper
  <area>
    <info width=38>
      <p>After you have placed your order with Window Shopper, you should...
      <ol>
        <li>Press the Esc key to leave the Order Panel.
        <li>Go to the receiving desk located at the front of the store.
          <p>Don't forget to bring your receipt!
        <li>Give the cashier the pink copy of your receipt.
        <li>Take your purchases home, and enjoy!
      </ol>
    </info>
  </area>
</panel>
```

The paragraph text follows the indentation of the preceding list item, like this:

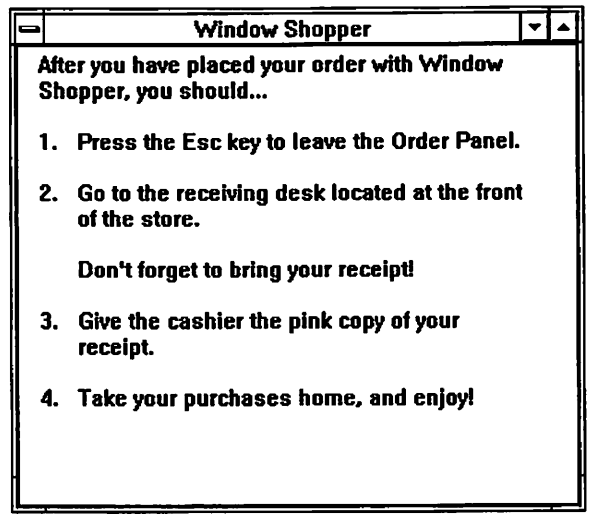


Figure 6-20. Nested Paragraph within a List

The List Part (LP) Tag

If you want to insert unindented text in a list, use the list part (LP) tag. The LP tag is useful for providing information about the list items that follow it.

We added a list part to the panel shown in Figure 6-20:

```
<panel name=winshop2 width=42 depth=18>Window Shopper
  <area>
    <info width=38>
      <p>After you have placed your order with Window Shopper, you should...
      <ol>
        <li>Press the Esc key to leave the Order Panel.
        <li>Go to the receiving desk located at the front of the store.
          <p>Don't forget to bring your receipt!
        <li>Give the cashier the pink copy of your receipt.
          <lp>Occasionally, the item you ordered won't be in stock.
          If this occurs, the cashier will be happy to delete
          the item from your order.
        <li>Take your purchases home, and enjoy!
      </ol>
    </info>
  </area>
</panel>
```

Here's the formatted result:

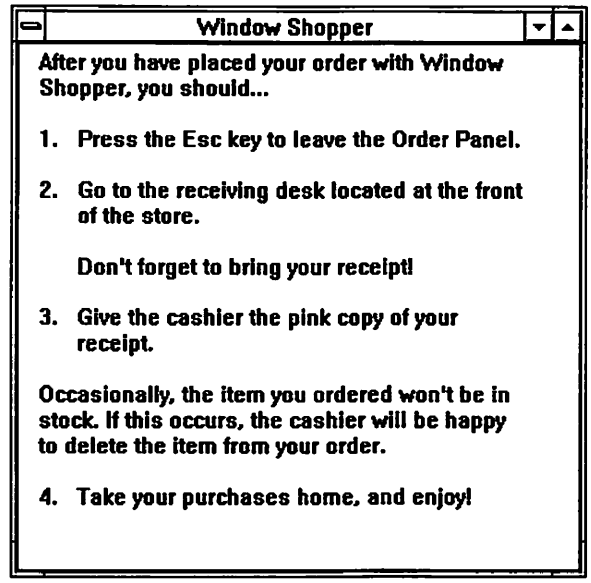


Figure 6-21. List Part

Nesting Lists within Lists

On pages 6-14 and 6-16 we showed you how to define levels of nested unordered and ordered lists. You can also nest different types of lists within other lists.

Here's an example of an unordered list nested within a definition list:

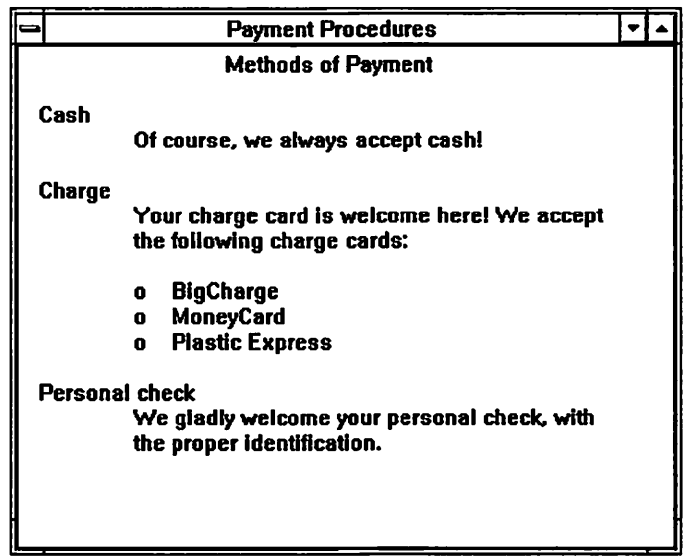


Figure 6-22. Nested Unordered List in a Definition List

Here's the markup we used to create the nested lists in Figure 6-22 on page 6-25:

```
<panel name=payment width=50 depth=20>Payment Procedures
  <area>
    <info width=46>
      <h1>Methods of Payment
      <dl break=all>
        <dt>Cash
        <dd>Of course, we always accept cash!
        <dt>Charge
        <dd>Your charge card is welcome here!
        We accept the following charge cards:
          <ul compact>
            <li>BigCharge
            <li>MoneyCard
            <li>Plastic Express
          </ul>
        <dt>Personal check
        <dd>We gladly welcome your personal check,
        with the proper identification.
      </dl>
    </info>
  </area>
</panel>
```

You can nest any type of list within another list. Remember, whenever you nest lists, be sure that you end each level with its proper end tag.

Alerting Users: Notes, Warnings, and Cautions

DTL provides you with tags that you can use to call the users' attention to certain passages in your text that warrant special attention. Whether it is noting a minor aspect of the application or alerting the user to the risk of possible damage to programs or data, you can alert the user appropriately.

In this section, we discuss the following tags:

- CAUTION
- NT
- NOTE
- WARNING.

All of these tags are language-sensitive, and depend on the /LANGUAGE option of the DTLC command when the source file is compiled. You can also use the /LANGUAGE option to change the appearance of these tags, for example, to change the case of the tag characters. See the description of the /LANGUAGE option in "DTLC" on page 13-2 for a complete description.

Notes

The NOTE and NT tags format as noted text. Use notes to emphasize minor points.

When you use either tag, you get the text "Note:" followed by two spaces before the text you specify. However, depending on which tag you use, the text is formatted differently.

There are no attributes associated with either tag.

The NOTE tag: If the text is a single paragraph, you use the NOTE tag. The text is formatted as an unindented block, like a paragraph. The NOTE tag does not require a matching end tag.

You use the NOTE tag like this:

```
<panel name=widgets width=42 depth=15>Widgets
  <area>
    <info width=38>
      <p>Choose the type of Widget you want to order by placing
      the cursor on the field and pressing Enter.
      <note>If the Widget you wish to order is not in stock, please
      refer to the "Back Order" panel to place an order.
    </info>
  </area>
</panel>
```

And this is how it formats:

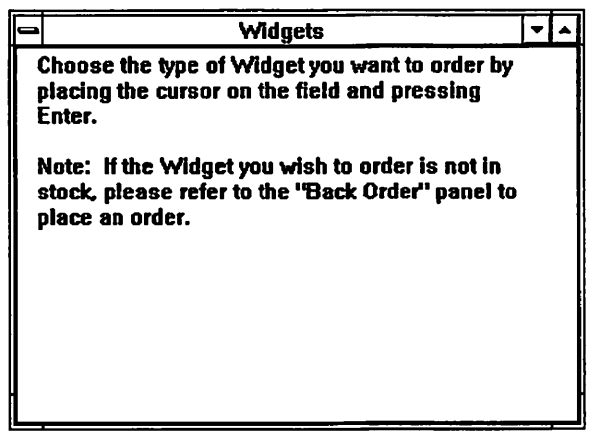


Figure 6-23. Note (NOTE tag)

The NT tag: If the note requires more than one paragraph, you use the NT tag. You use the P tag to add any additional paragraphs in the NT definition. Use the required end tag to end the NT definition.

Another difference between the NOTE and NT tag is that the NT tag indents the note text from the left panel margin.

In this example, the note is longer than one paragraph. We use the NT tag and its required end tag to define the note, and a P tag for each additional paragraph.

```
<panel name=widgets width=42 depth=20>Widgets
  <area>
    <info width=38>
      <p>Choose the type of Widget you want to order by placing
      the cursor on the field and pressing Enter.
      <nt>If the Widget you wish to order is not in stock, please
      refer to the "Back Order" panel to place an order.
        <p>Back-ordered Widgets usually arrive within three days.
      </nt>
      <p>If you want to order more than one Widget, specify the amount
      and press Enter.
    </info>
  </area>
</panel>
```

Notice that the P tag in the note is coded *before* the NT end tag, indicating that the second paragraph belongs in the note.

Here's how the panel looks now:

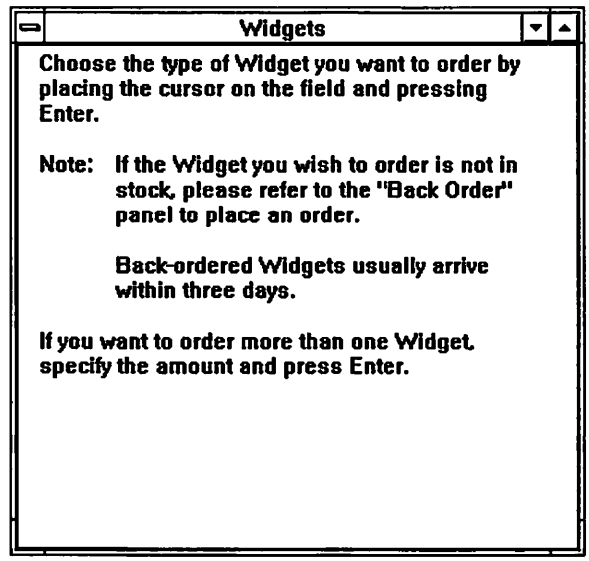


Figure 6-24. Note (NT tag)

As you can see, the the text of the NT tag is indented, as is the text of the P tag coded within the NT tag.

Warnings

Warning statements alert the user of a possible risk involved with a user action, or of existing error conditions.

You must immediately precede the WARNING tag with a P (paragraph) tag, LI (list item) tag, or LP (list part) tag. The warning statement formats with the term "Warning:" followed by two spaces before the text.

The WARNING tag has no associated attributes and requires a matching end tag.

Here's the markup for a warning statement that formats as a paragraph. We also nested an additional paragraph in the warning statement.

```
<panel name=addfile width=42 depth=15>Adding a File
  <area>
    <info width=38>
      <p>After you have made the desired changes
      to the file, press Enter to save the changes.
      <p><warning>Pressing Enter saves
      ALL changes made to the file.
      <p>You can cancel this operation
      by pressing the Esc=Cancel key.
      </warning>
    </info>
  </area>
</panel>
```

Here's the formatted result:

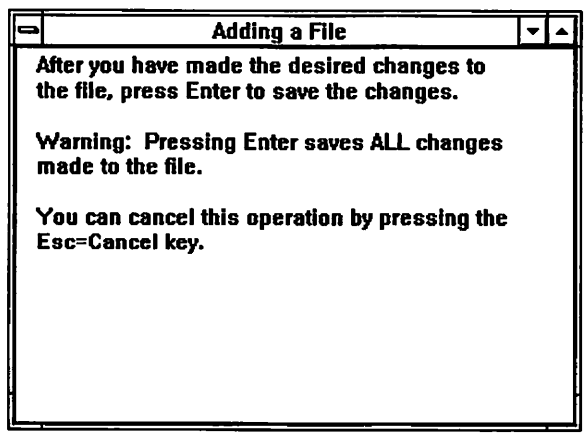


Figure 6-25. Warning

Cautions

Caution statements indicate the greatest degree of severity. Like the WARNING tag, the CAUTION tag has a required end tag, and must be preceded by a P (paragraph) tag, LI (list item) tag, or LP (list part) tag. The caution statement formats with the term "CAUTION:" followed by the text on the next line.

```
<panel name=delfile width=42 depth=20>Deleting a File
  <area>
    <info width=38>
      <p>To delete a file, type the filename in the
      "Delete this file" field and press Enter.
      <p>A message appears asking for verification.
      To continue the delete operation, press Enter.
      <p><caution>Verifying the delete operation
      permanently deletes the file from your records.
      There is no chance of recovery.
      </caution>
    </info>
  </area>
</panel>
```

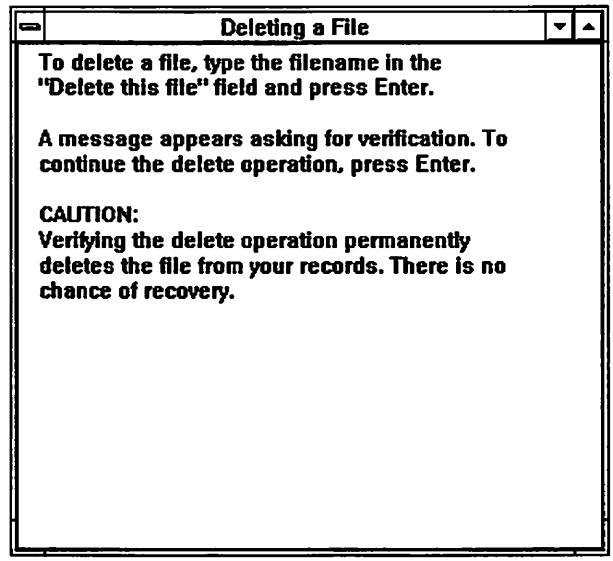


Figure 6-26. Caution

Using Information Regions with Other Panel Elements

Because they are so versatile, you can use information regions to complement the other elements of an application panel in many different ways. For example, you can use an information region to provide additional information for the fields on an application panel.

The information region in the following markup example uses a paragraph and a compact ordered list to tell the user how to interact with the panel fields. Figure 6-27 on page 6-32 shows the formatted result.

```
<panel name=appmnt width=50 depth=20>Make an Appointment
  <area>
    <info>
      <p>To schedule an appointment, you must choose one
      selection from each field.
      <ol compact>
        <li>Choose a day from the first field.
        <li>Choose a time slot from the second field.
        <li>After you have completed both fields, press
        Enter to log your appointment and leave the panel.
      </ol>
    </info>
    <divider type=solid>
    <region dir=horiz>
      <region>
      <selfld>Weekdays:
        <choice><m>Monday
        <choice><m>Tuesday
        <choice><m>Wednesday
        <choice>T<m>hursday
        <choice><m>Friday
      </selfld>
      </region>
      <divider gutter=5>
      <region>
        <selfld>Time:
          <choice>9:00
          <choice>10:00
          <choice>11:00
          <choice>12:00
          <choice>1:00
          <choice>2:00
          <choice>3:00
          <choice>4:00
        </selfld>
       </region>
    </region>
  </area>
</panel>
```

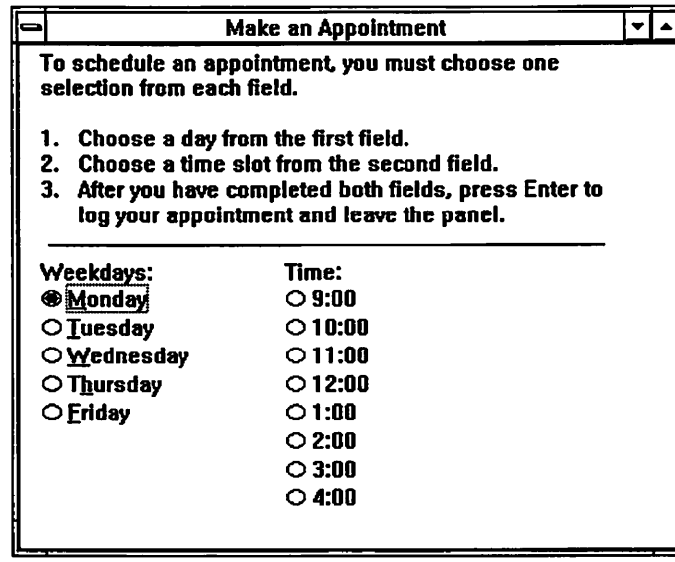


Figure 6-27. Information Region

Chapter 7. Help Panels

In this chapter, we show you how to use the DTL to define help panels that provide help to users while they are using a DM application. In addition to the HELP tag and its required end tag, which you use to define help panels, we discuss:

- The information region tags
- How to define reference phrases in help text
- How to define help index topics
- How to link help panels with application panels.

Defining Help Panels

The HELP tag and its required end tag define a help panel. The HELP start tag indicates the beginning of a help panel definition, and the HELP end tag closes the definition. All of the other tags that compose a help panel are coded between these two tags. You also use the HELP tag to define the panel title in the same way you code window title text with the PANEL tag, as tag content.

The HELP tag and its matching end tag look like this:

```
<help name=help01>Help Panel Title
</help>
```

In the above example, we added the required NAME attribute and value to the HELP start tag. Like the NAME attribute associated with the PANEL tag, the NAME value you assign must follow these conventions:

- It must not exceed eight characters in length.
- The first (or only) character must be alphabetic (upper- or lowercase).
- Any remaining characters can be either alphabetic or numeric.

The value you assign to NAME is the value that elements such as application panels, fields, and messages use to specify the help panel to provide help to the user. Reference phrases defined in help panels also use this value to link to the help associated with the reference phrase.

In addition to the HELP start and end tags, you must code an AREA start and end tag within every help panel for an application. This allows the user to scroll to portions of the panel that may not be visible during the initial display of the panel within the help text window.

Code the AREA definition within a help panel like this:

```
<help name=help01>Help Panel Title
  <area>
:
  </area>
</help>
```

Code the text you define for the help panel in the AREA definition.

The HELP tag also has a TUTORIAL attribute. This attribute allows you to specify the tutorial name that the application recognizes when a tutorial is requested. The tutorial name is passed to the application when the user requests the tutorial so that the appropriate processing can occur. Chapter 9, "Using Dialog Manager Help" in the *Dialog Manager Guide and Reference* provides a complete description of the TUTORIAL command.

When you specify a value for TUTORIAL, the **Tutorial** pull-down choice will display in the **Help** pull-down within the help window.

For example, this help panel uses the name "tutor01" as the parameter to the application-defined TUTORIAL command.

```
<help name=help01 tutorial=tutor01>Help Panel Title
  <area>
:
  </area>
</help>
```

Help Panel Information Regions

Some of the DTL tags and attributes that are supported in other environments are ignored by OS/2 DM when they are coded in help panels. The following list describes the attributes and tags ignored by the compiler in OS/2 DM applications.

HELP tag WIDTH attribute
: Like the PANEL tag, the HELP tag has WIDTH and DEPTH attributes that define the dimensions of the panel. These attributes are ignored by the compiler in OS/2 DM applications. Instead, help panels are displayed in a help window within the application main help window. This window is both sizable and scrollable by the user.

INFO tag WIDTH attribute
: The WIDTH attribute is ignored on INFO tags coded within help panels. Instead, the text of help panels flows dynamically to the width of the help window. The window can be sized by the user.

FIG tag FRAME and WIDTH attributes
: The FRAME and WIDTH attributes for the FIG tag are not supported for FIG tags coded in help panels.

Even though the compiler ignores the INFO WIDTH attribute in help panels, you must code an information region for the text of your help panels. Code the INFO tag and its matching end tag within the AREA definition, like this:

```
<help name=help01>Help Panel Title
  <area>
    <info>
:
    </info>
  </area>
</help>
```

The following section tells you how to define help panel text within information regions.

Defining Help Panel Text

The text you define for help panels cannot be modified by the user; it is for information purposes only. The only interaction the user can have with text in help panels is with defined reference phrases. (See "Defining Reference Phrases" on page 7-8 for more information about reference phrases.)

The following list contains the tags you can use to define help panel text within information regions.

- CAUTION
- DL (definition list)
- FIG (figure)
- FIGCAP (figure caption)
- Hn (heading)
- LINES
- NOTE
- NT (note)
- OL (ordered list)
- P (paragraph)
- PARML (parameter list)
- RP (reference phrase)
- SL (simple list)
- UL (unordered list)
- WARNING
- XMP (example).

For example, you use the P (paragraph) tag to define a paragraph of text on a help panel the same way you use it to define a paragraph on an application panel. In this section, we provide some examples of using the tags in the preceding list to define help panel text.

In this help panel markup, we use two paragraphs, an unordered list, a figure and figure caption to define the help text within the AREA definition. Figure 7-1 shows the formatted result.

```
<help name=olcathlp>Help for Online Catalog
  <area>
    <info>
       <p>The Online Catalog provides
       you with:
       <ul compact>
         <li>The book title
         <li>Catalog number
         <li>Page count
         <li>The author
         <li>A brief description
       </ul>
       <p>Here is an example:
       <fig>
       The Yellow Subroutine
       365 Pages              1234.56
       John-Paul Georgenringo

       A young band of British programmers
       embarks on a voyage across a perilous
       "sea" language in search of FORTRAN
       and fame.
         <figcap>Online Catalog Example
      </fig>
    </info>
  </area>
</help>
```

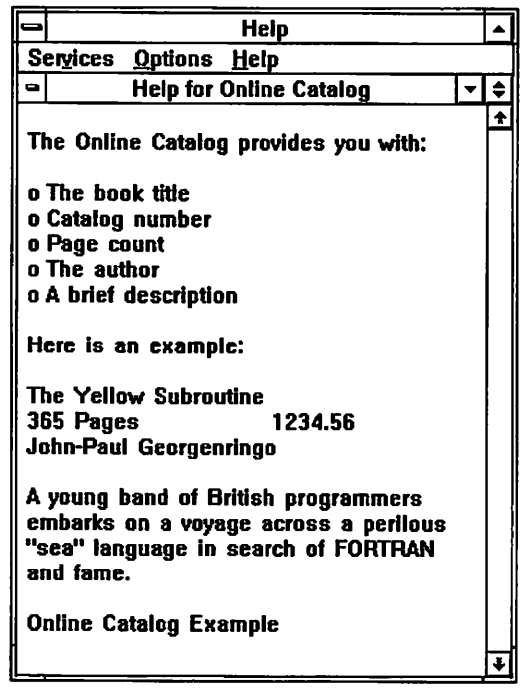


Figure 7-1. Help Panel (Basic Text)

This markup contains some paragraphs and headings. When the compiler processes help panels, it generates a table of contents using the help panel titles. For example, the HELP tag content, "Library Card Registration," will appear as an entry in the table of contents.

```
<help name=regishlp>Library Card Registration
  <area>
    <info>
       <h1>Library Card Registration
       <h2>New Registration
       <p>Complete the "New Registration" form.
       <h3>Full-time Resident
       <p>Complete all sections of the "New Registration" form,
       except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "New Registration" form.
       <h2>Card Renewal
       <p>To renew your current card, complete the
       "Renewal" form.
       <h3>Full-time Resident
       <p>Complete all sections of the "Renewal"
       form except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "Renewal" form.
    </info>
  </area>
</help>
```

Figure 7-2 shows the formatted result of the help panel. Figure 7-3 on page 7-6 shows the table of contents generated by the headings.

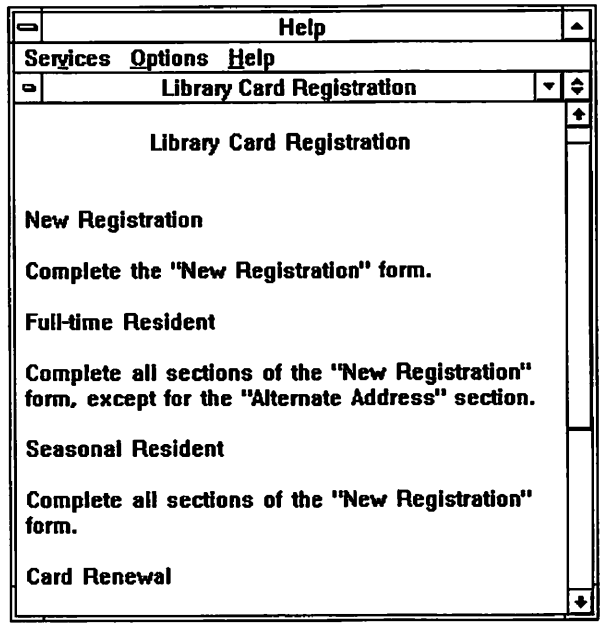


Figure 7-2. Help Panel (Headings)

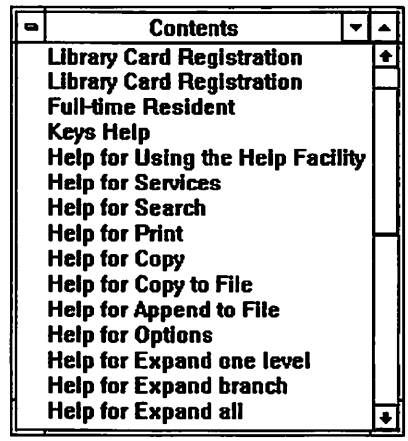


Figure 7-3. Table of Contents

This help panel contains a paragraph, a definition list, and a nested unordered list:

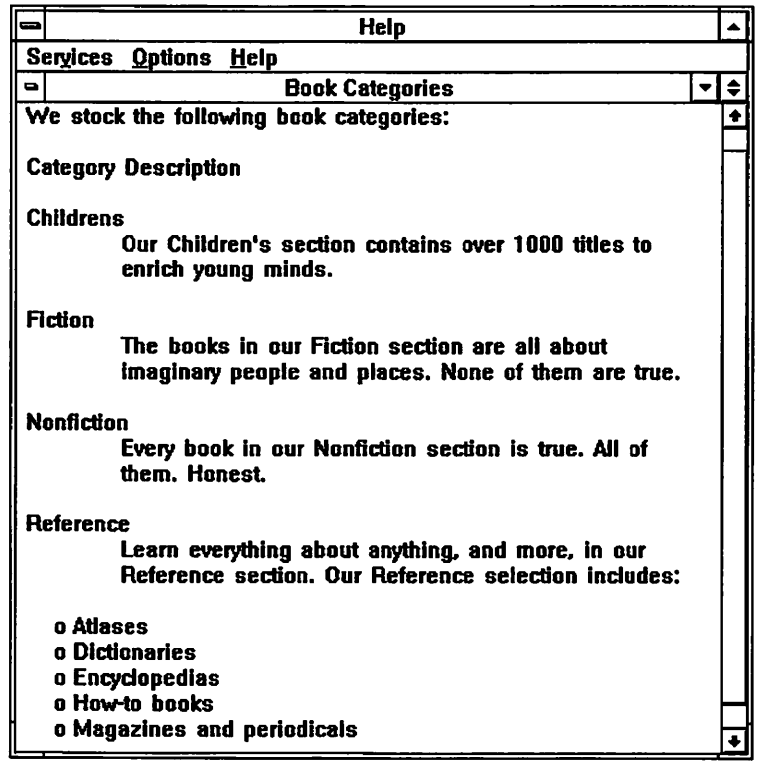


Figure 7-4. Help Panel (Definition List)

Here is the markup we used for Figure 7-4 on page 7-6:

```
<help name=category>Book Categories
  <area>
    <info>
      <p>We stock the following book categories:
      <dl break=all>
        <dthd>Category
        <ddhd>Description
        <dt>Childrens
        <dd>Our Children's section contains over 1000
        titles to enrich young minds.
        <dt>Fiction
        <dd>The books in our Fiction section are all
        about imaginary people and places.
        None of them are true.
        <dt>Nonfiction
        <dd>Every book in our Nonfiction section is true.
        All of them.
        Honest.
        <dt>Reference
        <dd>Learn everything about anything, and more, in
        our Reference section.
        Our Reference selection includes:
          <ul compact>
            <li>Atlases
            <li>Dictionaries
            <li>Encyclopedias
            <li>How-to books
            <li>Magazines and periodicals
          </ul>
      </dl>
    </info>
  </area>
</help>
```

Defining Reference Phrases

There is an additional information region tag that can only be used on help panels – the RP (reference phrase) tag. Use the RP tag to highlight words or phrases within help text that the user may be unfamiliar with, or that require explanation beyond the scope of the panel text. When the user selects the highlighted reference phrase and presses the Enter key, the help panel associated with the reference phrase appears within the help text window. This capability is referred to as *hypertext*.

To specify a reference phrase in your help text, code the RP tag immediately before the text you want to provide help for, and follow the text with the RP end tag. The required HELP attribute of the RP start tag specifies the name of the help panel associated with the reference phrase.

Note: You cannot code an RP tag within the text of a heading level one (H1) definition.

In this example, we added a reference phrase to the panel we created on page 7-5 that provides additional help to the user for the phrase "Full-time Resident."

```
<help name=regishlp>Library Card Registration
  <area>
    <info>
       <h1>Library Card Registration
       <h2>New Registration
       <p>Complete the "New Registration" form.
       <h3><rp help=ftres>Full-time Resident</rp>
       <p>Complete all sections of the "New Registration" form,
       except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "New Registration" form.
       <h2>Card Renewal
       <p>To renew your current card, complete the
       "Renewal" form.
       <h3>Full-time Resident
       <p>Complete all sections of the "Renewal"
       form except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "Renewal" form.
    </info>
  </area>
</help>
```

The value of the HELP attribute of the RP tag, *ftres*, matches the name of the help panel you define for this reference phrase.

When the help panel with the RP definition appears in the help window, the reference phrase is highlighted as shown:

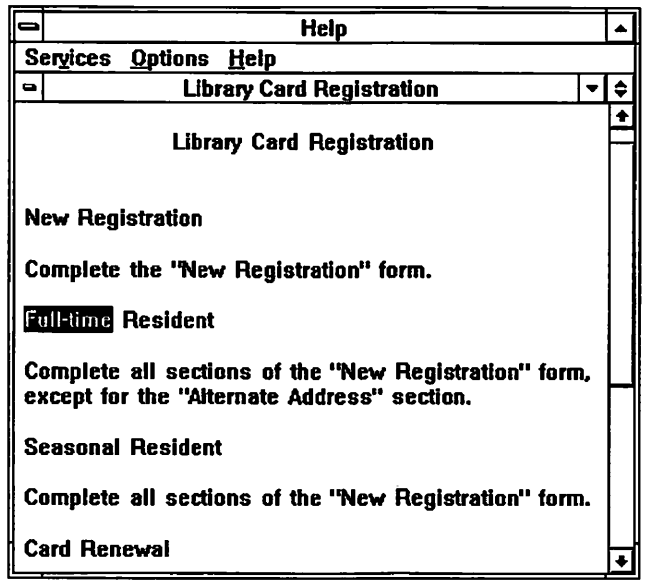


Figure 7-5. Reference Phrase

This highlighting indicates to the user that help is available for this topic. The user selects help by placing the cursor on the highlighted reference phrase and pressing Enter.

To show you what the help panel for this reference phrase might look like, we provided some markup for one. Figure 7-6 on page 7-10 shows the formatted result.

```
<help name=ftres>Full-time Resident
  <area>
    <info>
      <p>A full-time resident is a resident who establishes
      residency within the state for a minimum period of
      nine months during a given year.
      <p>This residency can be verified by any of the following:
      <sl compact>
        <li>State tax return
        <li>Utility receipts
        <li>Driver's license
      </sl>
    </info>
  </area>
</help>
```

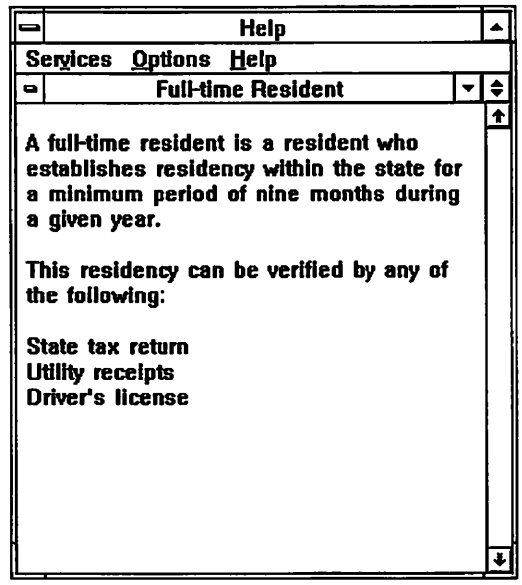


Figure 7-6. Reference Phrase Help Panel

Defining Help Index Topics and Synonyms

In addition to providing users with help for application panels and panel fields, you can define topics to be included in the help index. When the user is viewing the help index and selects a topic, the help panel the topic is associated with is displayed in the help window. The user can also access a help topic by typing in a synonym defined for the help topic in the search help index panel.

You define help topics and synonyms with the ITOP (index topic) and ISYN (index synonym) tags. You code the ITOP tag within the help panels that contain the help information you want to provide for the topics. Code both of these tags after the HELP start tag, before the AREA definition.

The ITOP (index topic) tag defines a topic for the help index. By adding the ITOP tag to the help panel, you specify the entry for the topic in the help index. The text of the ITOP tag defines the *index-topic-text* that appears in the help index.

You can code as many ITOP tags as you need in each help panel. However, you cannot use the same *index-topic-text* in more than one ITOP definition within the same help library. For example, the *index-topic-text* in this example, "Copying a document," is the text that will appear in the help index:

```
<help name=copyhlp>Help for Copying
:
  <itop roots='copy folder'>Copying a document
  <area>
    <info>
      <p>You must specify the following items to copy a file:
      <ul compact>
        <li>File name
        <li>Number of copies
          <p>If you do not specify an amount, the default amount is 1.
        <li>The bin number the copy or copies are to be sent to.
      </ul>
    </info>
  </area>
</help>
```

The required ROOTS attribute of the ITOP tag specifies the *root-word-list* for the index topic (in this case, *copy folder*, is the *root-word-list*). The words you specify for the *root-word-list* serve as the link between the ITOP tag and the associated ISYN tags. Because a *root-word-list* can link to multiple ISYN tags, you must enclose the *root-word-list* in single quotes, and place blanks between each of the *root-words*. This list is not meant to be translated, it serves only to link the ITOP definition with associated ISYN definitions.

To continue with the preceding example, we'll add an ISYN definition that links to the ITOP definition we have already established. The ISYN tag has a required ROOT attribute that you use to specify a root word for the synonyms you define as tag text for the ISYN tag. This is the value that links to the *root-word-list* defined in the ITOP tag. For example, if we specify *copy* as the *root-word* value for ISYN, we would define variations or synonyms for *copy* as ISYN tag text.

```
<help name=copyhlp>Help for Copying
  <isyn root=copy>copy copying duplicate duplicating
  <itop roots='copy folder'>Copying a document
  <area>
    <info>
      <p>You must specify the following items to copy a file:
      <ul compact>
        <li>File name
        <li>Number of copies
          <p>If you do not specify an amount, the default amount is 1.
        <li>The bin number the copy or copies are to be sent to.
      </ul>
    </info>
  </area>
</help>
```

The *root-word* serves as the other side of the link between the ISYN and ITOP tags. The synonyms you define for the *root-word*, in this case "copy, copying, duplicate, and duplicating," can be entered by the user in the index search panel when they are searching for a topic. Think of the *root-word* as an entry in a thesaurus, and the synonyms as a list of the corresponding synonyms to the entry. Similarly, the *root-word-list* defined in the ITOP tag is a list of all of the thesaurus entries which map to the help panel they are defined within.

To complete the preceding example, we need to add another ISYN definition that defines the second link in the ITOP *root-word-list*. We'll also define the synonyms for the *root-word*.

```
<help name=copyhlp>Help for Copying
  <isyn root=copy>copy copying duplicate duplicating
  <isyn root=folder>folder folders document documents file files
  <itop roots='copy folder'>Copying a document
  <area>
    <info>
      <p>You must specify the following items to copy a file:
      <ul compact>
        <li>File name
        <li>Number of copies
          <p>If you do not specify an amount, the default amount is 1.
        <li>The bin number the copy or copies are to be sent to.
      </ul>
    </info>
  </area>
</help>
```

When the user performs a search on any of the synonyms defined in the ISYN tags, the "Copying a document" index entry would be found.

As we stated earlier, you can define multiple index topic entries in a single help panel. And, if necessary, these ITOP definitions can all link to common ISYN definitions. In the following example, we defined index topics and synonyms for the help panel we defined on page 7-7.

```
<help name=category>Book Categories
  <isyn root=book>book title
  <isyn root=category>category type
  <isyn root=children>childrens youth beginning
  <isyn root=fiction>fiction novel play mystery
  <isyn root=nonfiction>nonfiction biography
  <isyn root=reference>reference atlas dictionary encyclopedia
  <itop roots='book category'>Book categories
  <itop roots='book children'>Children's books
  <itop roots='book fiction'>Fiction books
  <itop roots='book nonfiction'>Nonfiction books
  <itop roots='book reference'>Reference books
  <area>
    <info>
      <p>We stock the following book categories:
      <dl break=all>
        <dthd>Category
        <ddhd>Description
        <dt>Childrens
        <dd>Our Children's section contains over 1000
        titles to enrich young minds.
        <dt>Fiction
        <dd>The books in our Fiction section are all
        about imaginary people and places.
        None of them are true.
        <dt>Nonfiction
        <dd>Every book in our Nonfiction section is true.
        All of them.
        Honest.
        <dt>Reference
        <dd>Learn everything about anything, and more, in
        our Reference section.
        Our Reference selection includes:
          <ul compact>
            <li>Atlases
            <li>Dictionaries
            <li>Encyclopedias
            <li>How-to books
            <li>Magazines and periodicals
          </ul>
      </dl>
    </info>
  </area>
</help>
```

Defining Command Help

You can identify a help panel as containing help information for an application command by adding an ICMD (index command) tag to the panel markup. Users can select help for commands either by placing the cursor in the command entry field of an application panel and requesting help, or by selecting a command listing from a list panel, which is built from all of the ICMD tags you code within help panels.

To illustrate how ICMD tags generate help for commands, we'll add Index command definitions to the "Library Card Registration" panel we developed on page 7-5.

```
<help name=regishlp>Library Card Registration
  <icmd>Register
  <icmd>Renewal
  <area>
    <info>
       <h1>Library Card Registration
       <h2>New Registration
       <p>Complete the "New Registration" form.
       <h3>Full-time Resident
       <p>Complete all sections of the "New Registration" form,
       except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "New Registration" form.
       <h2>Card Renewal
       <p>To renew your current card, complete the
       "Renewal" form.
       <h3>Full-time Resident
       <p>Complete all sections of the "Renewal"
       form except for the "Alternate Address" section.
       <h3>Seasonal Resident
       <p>Complete all sections of the "Renewal" form.
    </info>
  </area>
</help>
```

Always code ICMD tags *before* the AREA tag in a help panel.

The help panel for this markup appears when users request help for "register" or "renewal" from the command entry field of an application panel, or when they select the command from the list of all command help panels.

Providing Help for Application Panels

To link the help panels you create for your applications to the application element you want to provide help for, you use the HELP attribute of the tag that defines the application element. The value you assign the HELP attribute is the NAME value of the help panel that provides the help. Using the HELP attribute to specify the help panel, you can provide help for:

- Application panels
- Panel fields
- Action bar and pull-down choices
- Messages.

The following table shows you which DTL tags contain the HELP attribute and the panel or panel element they represent:

Tag	Purpose
ABC	Specifies help for an action bar choice.
CHOICE	Specifies help for a selection field choice.
DTAFLD	Specifies help for a data entry field.
LSTCOL	Specifies help for a list column.
LSTFLD	Specifies help for a list field.
MSG	Specifies help for a message.
PANDEF	Specifies help for application panels using the panel default.
PANEL	Specifies help for an application panel.
PDC	Specifies help for a pull-down choice.
SELFLD	Specifies help for a selection field.
SELLST	Specifies help for a selection list.
UC	Specifies help for a user control.

Keys Help

To provide help information for the function keys defined for an application, you should define a keys help panel. This panel can, for example, contain a list of the available keys and their descriptions.

One method for doing this is to use a definition list, like this:

```
<help name=khelp>Keys Help
  <area>
    <info>
      <p>The following function keys are available:
      <dl>
        <dt>F1
        <dd>which allows you to request help on a panel, panel field,
        or message.
        <dt>F3
        <dd>which allows you to exit the current panel.
      </dl>
    </info>
  </area>
</help>
```

The preceding markup formats like this:

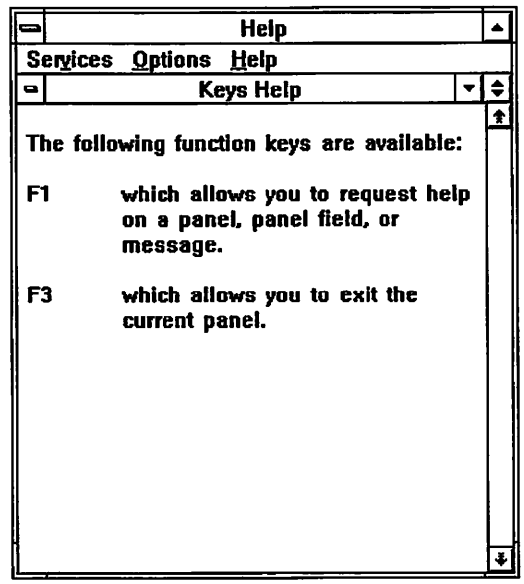


Figure 7-7. Keys Help Panel

For more information on keys help, refer to the *Dialog Manager Guide and Reference*.

Chapter 8. Messages

You use messages to communicate unsolicited information to users; that is, information that you, the application developer, believe they need to know. Typically, this would be information regarding user actions, status, or problems that need correction.

The Dialog Tag Language provides you with the means of defining and storing messages for Dialog Manager applications. After you define them, you use the Dialog Manager to handle the communication and display of messages in the application. When the application calls for a message to be displayed, the Dialog Manager places it within a pop-up window, known as a *message pop-up*.

Additionally, messages are defined according to their purpose and severity. The three types of messages you can define are:

information To provide information about a user-requested action.

warning To provide information about conditions the user may need to be aware of.

action To alert the user to an exception condition that requires a response from the user to correct the situation.

This chapter tells you how to define messages and how to store them.

Defining Messages

You store the messages you define for your application within *message members*. Each message member can contain up to ten messages. The DTL compiler stores the message members within a DTL library file for the application. The DTL tags you use to define messages and message members are:

MSGMBR Which, with its required end tag, defines a message member.

MSG Which defines a message within a message member. The text of the MSG tag is the text that appears as the message. Each message can be up to 512 bytes in length after variable substitution. "Variable Substitution" on page 8-4 describes variable substitution in messages.

You assign an identifier to each message within a message member. The message identifier is composed of two required attribute values: the NAME attribute value of the MSGMBR tag and the SUFFIX attribute value for the MSG tag.

The NAME attribute you specify for the MSGMBR tag must consist of three uppercase or lowercase alphabetic characters, one alphabetic character (A–F only), and two numeric characters. For example:

```
<msgmbr name=maia00>
```

The SUFFIX attribute values for each of the MSG tags you code within a MSGMBR definition must have a one-digit numeric character (0–9), and each of the values must be unique (a message suffix cannot be defined twice in a message member).

Consequently, a message member can contain a maximum of ten messages, as in this example:

```
<msgmbr name=maia00>
  <msg suffix=0>You cannot type a number in the Name field.
  <msg suffix=1>Please include your first name in the Name field.
  <msg suffix=2>Unrecognized character in Name field.  Please correct.
  <msg suffix=3>Unrecognized character in Address field.  Please correct.
  <msg suffix=4>You cannot type a number in the City field.
  <msg suffix=5>Unrecognized character in City field.  Please correct.
  <msg suffix=6>You cannot type a number in the State field.
  <msg suffix=7>You must type two letters in the State field.
  <msg suffix=8>The Zip code exceeds the maximum length.
  <msg suffix=9>You cannot type an alphabetic character in the Zip field.
</msgmbr>
```

The value of the MSG SUFFIX attribute, when added to the MSGMBR NAME value, forms the message identifier for that message. For example, the message identifier for the message: "The Zip code exceeds the maximum length," is *maia008*. You assign this value to the MSG attribute of the panel element the message is associated with, and it appears as a result of field validation.

The message identifiers for all of the messages in the *maia00* message member are as follows:

```
maia000
maia001
maia002
maia003
maia004
maia005
maia006
maia007
maia008
maia009
```

In addition to SUFFIX, the MSG tag has an optional HELP attribute that allows you to define a help panel for a message. For more information on defining help panels, see Chapter 7, "Help Panels" on page 7-1.

Specifying Message Types

You can also specify the severity of a message with the MSGTYPE attribute of the MSG tag. The severity you assign a message determines how the alarm tones are used when the message is displayed. The Dialog Manager accepts one of three values for the MSGTYPE attribute: INFO (the default value), WARNING, or ACTION.

Information Messages: Use the default value INFO when you want to provide the user with feedback about the state of the application.

```
<msgmbr name=orda00>
  <msg suffix=0>Your order is being processed.  Please wait...
</msgmbr>
```

Warning Messages: Warning messages tell users that a potentially undesirable situation could occur. Users only need to respond to the message to continue, although corrective action may be required later.

```
<msgmbr name=orda00>
  <msg suffix=0>Your order is being processed.  Please wait...
  <msg suffix=1 msgtype=warning>Your request for the engraving
  option is not valid.
  Please check your request, and correct it if necessary.
</msgmbr>
```

Action Messages: Action messages represent the highest degree of severity. They tell users about exception conditions that require a response. The user must respond with a specific action to continue with the application.

```
<msgmbr name=orda00>
  <msg suffix=0>Your order is being received.  Please wait...
  <msg suffix=1 msgtype=warning>Your request for the engraving
  option is not valid.
  Please check your request, and correct it if necessary.
  <msg suffix=2 msgtype=action>The data you have entered is
  incorrect.
  Please reenter the data.
</msgmbr>
```

Regardless of the severity of the message, the Dialog Manager displays all messages in message pop-ups. Additionally, the Dialog Manager adds an audible alarm to warning and action messages. For more information on message display and positioning, refer to the *Dialog Manager Guide and Reference*.

Assigning Messages

Each of the panel elements that you can assign messages to have optional MSG attributes that you use to specify a message identifier. This indicates to the Dialog Manager the message to be displayed if the user's input doesn't pass any of the validation checks for the panel element.

The following list contains the DTL tags that have MSG attributes associated with them, and describes the conditions for usage for each one.

CHECKL Use the MSG attribute of the CHECKL tag to specify a message the Dialog Manager displays when the user's input fails any of the validation checks defined for the check list.

DTAFLD Use the MSG attribute of the DTAFLD tag to specify a message the Dialog Manager displays when the user does not complete a required field. Thus, you can only assign a message to a data field when the REQUIRED attribute has a value of YES.

LSTCOL Use the MSG attribute of the LSTCOL tag to specify a message the Dialog Manager displays when the user does not complete a required entry. Consequently, you can only assign a message to a list column when the REQUIRED attribute has a value of YES.

VARCLASS Use the MSG attribute of the VARCLASS tag to specify a message the Dialog Manager displays when the user's input fails the validity check defined by the VARCLASS TYPE attribute.

XLATL Use the MSG attribute of the XLATL tag to specify a message the Dialog Manager displays when the user's input fails a specified translation.

The Dialog Manager displays a default message for each of the tags listed above if you do not specify the MSG attribute.

To show you how messages are associated with panel elements, we'll define a data field that requires input from the user. We'll also define a message member that contains the warning message the Dialog Manager displays if the user does not complete the field. Figure 8-1 shows the displayed panel and message.

```
<!doctype dm system>

<varclass name=namecls type='char 30'>

<varlist>
  <vardcl name=name varclass=namecls>
</varlist>

<msgmbr name=ordb00>
  <msg suffix=0 msgtype=warning>You must type your name in the Name field.
</msgmbr>

<panel name=dmm001 width=40 depth=15>Catalog Order Form
⋮
 <dtafld datavar=name required=yes msg=ordb000>Name

⋮
</panel>
```

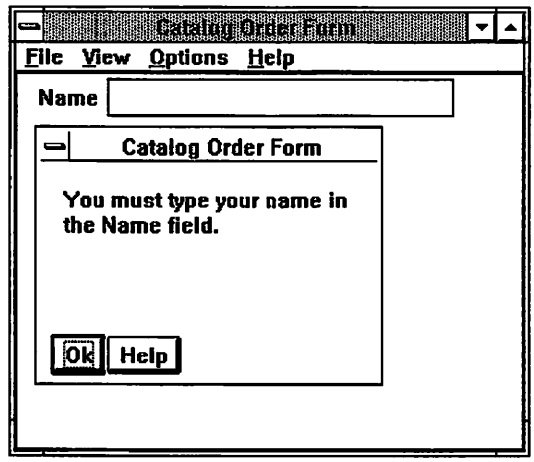


Figure 8-1. Data Field and Message

Variable Substitution

You can specify a variable in the text of a message using the VARSUB (variable substitution) tag. The value you specify with VARSUB can be derived from a declared variable or from a value the user entered in a field. When the message is displayed, the Dialog Manager inserts the current value of the variable into the text of the displayed message.

You code the VARSUB tag within the text of the message where you want to substitute text. You must also specify one of the following VARSUB attributes:

VAR To specify the name of a declared variable whose value is substituted as message text.

FIELD To specify that the data entered by the user in the current field is substituted as message text.

Substituting a Variable Value

In the following example, we used two variable substitutions in the text of the message "msga001." The first VARSUB specifies the variable declaration, *invvar*, which provides an invoice number in the message. The second VARSUB specifies the variable declaration, *datevar*, which provides a date in the message.

```
<!doctype dm system>

<varclass name=invoices type='char 10'>
<varclass name=updates  type=idate>

<varlist>
  <vardcl name=invvar varclass=invoices>
  <vardcl name=datevar varclass=updates>
</varlist>

<msgmbr name=msga00>
  <msg suffix=0>Your request is being processed.
  <msg suffix=1>The invoice number you have requested,
  <varsub var=invvar>, was last updated on
  <varsub var=datevar>.
</msgmbr>
```

Substituting a Field Value

The variable substitution in this message specifies a user-entered value in the **Age** data field. If the value the user enters fails the validation checks for the data field, the Dialog Manager displays the value in the message.

```
<!doctype dm system>

<varclass name=agecls type='char 3'>
  <checkl msg=agea000>
    <checki type=range parm1=1 parm2=120>
  </checkl>

<varlist>
  <vardcl name=age varclass=agecls>
</varlist>

<panel name=agefield>
⋮
 <dtafld datavar=age usage=in entwidth=3>Age
⋮
</panel>

<msgmbr name=agea00>
  <msg suffix=0>The value you entered, <varsub field>,
  is not acceptable.
</msgmbr>
```

Assuming the entered value was "125," the message would appear as shown in Figure 8-2:

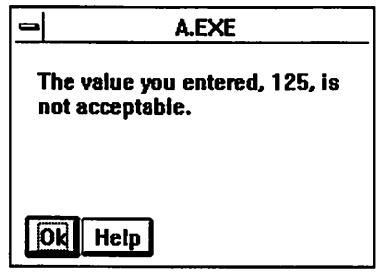


Figure 8-2. Variable Substitution

Chapter 9. The Application Command Table

In addition to the system commands in the Dialog Manager command table, DTL provides you with a way to define and store commands that are specific to your application. You can also define commands that override the DM-provided commands. You define and store these commands within a *command table* for your application. These application-specific commands define the responses to commands entered by the user in the command entry field and commands linked to selection field choices, pull-down choices, and key mapping lists.

For more information on the DM-provided commands, refer to the *Dialog Manager Guide and Reference*.

You can define only one command table for an application. You can define the application command table within any .GML file, and Dialog Manager locates the command table using the defined application identifier.

Defining the Application Command Table

The tags you use to define an application command table are:

CMDTBL To begin the definition of an application command table. The required end tag ends the definition.

CMD To define a command within an application command table. You code the CMD tags within a CMDTBL definition (between the start and end tags).

CMDACT To define the action taken by the Dialog Manager when a user executes a command. You code the CMDACT tag following the command (CMD) it is associated with.

There are two attributes associated with the CMDTBL tag: the required APPLID (*application-identifier*) attribute and the optional NOMATCH attribute. The Dialog Manager uses the value you assign to the APPLID attribute to identify the command table. The value you assign to the NOMATCH attribute determines how the Dialog Manager responds to commands that are not defined in the application command table or that are not one of the DM-defined commands. The valid values for NOMATCH are ERROR and APPLCMD.

The value you assign as an application identifier can have a maximum of four characters, and the first character must be alphabetic. Any remaining characters can be either alphabetic or numeric. Additionally, the Dialog Manager reserves the application identifier ISPx, where x is any character including the space character. Do not use any of these for an APPLID value.

DTL uses the application identifier as a prefix to the string CMDS to form the name of the command table library. For example, the APPLID value, *demo*, forms the prefix for the application command table name DEMOCMDS.DTL.

When a user enters a command in a command entry field or through an action bar, selection field, or function key, the Dialog Manager searches the application command table for the command and performs the action you define for the command when the command is found. If the command is not found in the application command table, the Dialog Manager searches the list of DM-provided commands, and, if found, performs the action for that command.

If the command is not one of the DM-provided commands, one of two actions will occur based on the value you specify for the NOMATCH attribute of the CMDTBL tag:

- When NOMATCH=ERROR (the default value), the Dialog Manager displays an error message.
- When NOMATCH=APPLCMD, the Dialog Manager updates the ZCMD system variable with the command name and parameters. Following that, the Dialog Manager returns to the application to allow the command to be processed. It is up to the application to display an error message if it determines that the command is not valid.

If you do not define a command table for an application, the Dialog Manager searches the list of DM-provided commands. If the command is not one of the DM-provided commands, an error message is issued.

Use the CMD tag to define each of the commands within the application command table. The CMD tag has a required NAME attribute that you use to identify the *internal-command-name* for the command. The value you assign as an *internal-command-name* must not exceed eight characters, and the first character must be alphabetic. Any remaining characters can be either alphabetic or numeric.

Optionally, you can assign a translatable string (tag text) to a CMD definition in addition to the required *internal-command-name*. This string represents the *external-command-name* for the command. The *external-command-name* is used as the search criteria when the user enters a command from the command entry field. This means that any application-specific commands entered by the user in the command entry field must be defined with both *internal-* and *external-command-names*.

The following markup example shows a source file that contains an application command table, an action bar, and a key mapping list. The command table contains commands that are mapped to the values specified in the RUN attributes of the ACTION tags and the CMD attributes of the KEYI tags.

```
<!doctype dm system>

<cmdtbl applid=brws>
  <cmd name=quit>quit
    <cmdact action=...>
  <cmd name=send>send
    <cmdact action=...>
  <cmd name=file>file
    <cmdact action=...>
  <cmd name=edit>edit
    <cmdact action=...>
  <cmd name=restore>restore
    <cmdact action=...>
</cmdtbl>

<keyl name=panlkeys>
  <keyi key=f3 state=alt cmd=quit>
  <keyi key=f6 cmd=send>
  <keyi key=f4 state=alt cmd=file>
  <keyi key=f2 state=alt cmd=edit>
  <keyi key=f7 cmd=restore>
</keyl>

<panel...>
  <ab>
    <abc><m>Actions
      <pdc><m>Delete
        <action run=quit>
      <pdc><m>Send
        <action run=send>
      <pdc><m>File
        <action run=file>
      <pdc><m>Edit
        <action run=edit>
      <pdc><m>Restore
        <action run=restore>
  </ab>
:
</panel>
```

Because the Dialog Manager provides the ENTER and CANCEL commands, these commands are not defined within the application command table. When the ENTER or CANCEL commands are entered, the Dialog Manager processes the commands from the list of DM-provided commands.

You can, if you wish, override any of the DM-provided commands by defining your own commands in the application command table. To do this, specify the system command name (as it is defined by the Dialog Manager) as the *internal-command-name* of the CMD definition. However, keep in mind that your definition replaces the DM-provided definition for processing the command.

Specifying Command Actions

You must specify a CMDACT tag for each of the CMD tags you define within an application command table so that the Dialog Manager can process the commands. You use the CMDACT tag to define the action taken for the command when it is selected. Code the CMDACT tag immediately after the CMD tag it is associated with. There is no required end tag for the CMDACT tag.

The CMDACT tag has two attributes: ACTION and CLASS. You must code one of these attributes for each CMDACT definition. The ACTION attribute allows you to specify a DM-processed action for the command. The CLASS attribute allows you to specify an application-coded Presentation Manager class that processes the command.

The ACTION attribute

You use the ACTION attribute of the CMDACT tag to specify a DM-defined action for the command. You can specify one of the following Dialog Manager command actions for each CMDACT definition:

- BACKWARD
- CANCEL
- CHGDEFS
- ENTER
- EXHELP
- EXIT
- FKA
- FORWARD
- HELP
- HELPHELP
- INDEX
- KEYS
- LEFT
- PANELID
- RETRIEVE
- RIGHT.

For a complete description of each of the Dialog Manager commands, see "CMDACT (Command Action)" on page 15-37. In addition, Chapter 15, "Dialog Manager Commands and Descriptions" in the *Dialog Manager Guide and Reference* contains descriptions of the Dialog Manager commands.

We'll specify EXIT as the command action for the QUIT command defined in the application command table on page 9-2.

```
<cmdtbl applid=brws>
 <cmd name=quit>quit
  <cmdact action=EXIT>
 <cmd name=send>send
  <cmdact action=...>
 <cmd name=file>file
  <cmdact action=...>
 <cmd name=edit>edit
  <cmdact action=...>
 <cmd name=restore>restore
  <cmdact action=...>
</cmdtbl>
```

In addition to the Dialog Manager commands, you can assign one of the following values to the ACTION attribute:

ALIAS To allow the user to express the command in an alternative manner, such as using QUIT as an alias for EXIT.

PASSTHRU To pass the command to the application. The *internal-command-name* and any command parameters are passed to the dialog in the Dialog Manager ZCMD system variable.

SETVERB To pass the command to the application. The *internal-command-name* is passed to the dialog in the ZVERB system variable, and the parameters (if any) are passed to the dialog in the Dialog Manager ZCMD system variable.

The ALIAS command action provides you with a way to define synonyms for commands. The *internal-command-name* you define for the ALIAS defines the synonym for the command it is associated with. Additionally, you must enclose the ALIAS value, the *internal-command-name* for the alias, and any parameters within quotes.

When you define an ALIAS command action, you must code the CMDACT tag for the ALIAS before the command the ALIAS represents. Because the Dialog Manager searches the application-defined commands sequentially, and then the DM-provided commands, it must locate the ALIAS definition before the aliased command.

In this example, we've added the commands UP and DOWN to the application command table. The command actions defined for the commands specify the Dialog Manager commands BACKWARD and FORWARD as aliases for the commands. Because the BACKWARD and FORWARD commands are provided by the Dialog Manager, we do not need to define them in the application command table. The Dialog Manager will locate the aliases before the DM-provided commands they reference.

```
<cmdtbl applid=brws>
  <cmd name=quit>quit
    <cmdact action=EXIT>
  <cmd name=send>send
    <cmdact action=...>
  <cmd name=file>file
    <cmdact action=...>
  <cmd name=edit>edit
    <cmdact action=...>
  <cmd name=restore>restore
    <cmdact action=...>
  <cmd name=up>
    <cmdact action='alias backward'>
  <cmd name=down>
    <cmdact action='alias forward'>
</cmdtbl>
```

We'll specify PASSTHRU as the command action for the SEND, EDIT, and RESTORE commands in the command table. PASSTHRU passes the commands and any parameters associated with the commands to the dialog in the ZCMD system variable.

```
<cmdtbl applid=brws>
  <cmd name=quit>quit
    <cmdact action=EXIT>
  <cmd name=send>send
    <cmdact action=passthru>
  <cmd name=file>file
    <cmdact action=...>
  <cmd name=edit>edit
    <cmdact action=passthru>
  <cmd name=restore>restore
    <cmdact action=passthru>
  <cmd name=up>
    <cmdact action='alias backward'>
  <cmd name=down>
    <cmdact action='alias forward'>
</cmdtbl>
```

You can also use the ACTION attribute to specify a variable that you declare in a variable list in your application. The Dialog Manager retrieves the variable value at run time when the command associated with the command action is processed.

The value of the variable must be one of the DM-defined command actions listed on page 9-4. You specify the variable using the % notation in the ACTION value.

We'll specify a declared variable as a command action for the FILE command. When the user issues the FILE command, the Dialog Manager obtains the dynamic variable from the variable pool to determine the action to be taken.

```
<!doctype dm system>

<varclass name=string type='char 8'>

<varlist>
  <vardcl name=update varclass=string>
</varlist>

<cmdtbl applid=brws>
  <cmd name=quit>quit
    <cmdact action=EXIT>
  <cmd name=send>send
    <cmdact action=passthru>
  <cmd name=file>file
    <cmdact action='%update'>
  <cmd name=edit>edit
    <cmdact action=passthru>
  <cmd name=restore>restore
    <cmdact action=passthru>
  <cmd name=up>
    <cmdact action='alias backward'>
  <cmd name=down>
    <cmdact action='alias forward'>
</cmdtbl>
```

The CLASS Attribute

Use the CLASS attribute to specify a Presentation Manager class name that performs an action user exit for the command. For a detailed description of user exits, refer to Chapter 13, "Additional Features of the Dialog Manager" in the *Dialog Manager Guide and Reference.*

We'll redefine a command action for the EDIT command that calls the Presentation Manager class "UserCmdActionClass" when the EDIT command is invoked. In addition, we'll provide a dynamic parameter that is also passed to the "UserCmdActionClass" exit routine.

```
<cmdtbl applid=brws>
  <cmd name=quit>quit
    <cmdact action=EXIT>
  <cmd name=send>send
    <cmdact action=passthru>
  <cmd name=file>file
    <cmdact action='%update'>
  <cmd name=edit>edit
    <cmdact class=UserCmdActionClass parm='%file'>
  <cmd name=restore>restore
    <cmdact action=passthru>
  <cmd name=up>
    <cmdact action='alias backward'>
  <cmd name=down>
    <cmdact action='alias forward'>
</cmdtbl>
```

Truncating Commands

You can specify a minimum command name of two characters for a command with the T (truncation) tag. The user can issue a truncated command in the command area by entering the minimum number of characters you specify for the command.

To specify truncation for a command, you code the T tag within the external command name of the command.

For example, to specify "qu" as the minimum command for the QUIT command, you add the T tag to the external command name, like this:

```
<cmdtbl applid=brws>
  <cmd name=quit>qu<t>it
    <cmdact action=EXIT>
:
</cmdtbl>
```

The T tag follows the characters you specify as the minimum command.

With this truncation, the user can issue the QUIT command by typing the command in one of the following ways:

```
qu
qui
quit
```

However, you should be careful when adding truncations that duplicate other truncations to a command table. For example, these two truncations define minimum commands ("co") that are identical:

```
<cmdtbl applid=brws>
  <cmd name=comp>co<t>mpare
    <cmdact action=passthru>
  <cmd name=copy>co<t>py
    <cmdact action=passthru>
</cmdtbl>
```

The preceding definition would cause the compiler to issue a warning message.

To avoid this type of duplication, place the T tag appropriately in the CMD tag content. The above duplication can be avoided by coding the truncations as follows:

```
<cmdtbl applid=brws>
  <cmd name=comp>com<t>pare
    <cmdact action=passthru>
  <cmd name=copy>cop<t>y
    <cmdact action=passthru>
</cmdtbl>
```

Another way we could avoid duplicating the "compare" and "copy" truncations is to change only one of the truncations, like this:

```
<cmdtbl applid=brws>
  <cmd name=comp>co<t>mpare
    <cmdact action=passthru>
  <cmd name=copy>cop<t>y
    <cmdact action=passthru>
</cmdtbl>
```

Chapter 10. Defining Key Mapping Lists

Every application panel has keys that map to the actions that are valid for the panel. You define these key assignments within key mapping lists. The key assignments map a key or a combination of keys to a command defined within the application command table or a DM-provided command. You use the KEYLIST attribute of the PANEL or PANDEF tag to name the key mapping list for an application panel.

The tags you use to define key mapping lists are:

KEYL To define a key mapping list. The required end tag ends the key mapping list definition.

KEYI To define a key assignment and specify the command the Dialog Manager processes when the user presses the key, and specify the label for the key if it is displayed in the function key area.

You can code multiple KEYI tags within a KEYL definition. You code a KEYI tag for each key that is active within an application panel.

Assigning Keys and Actions

Each KEYI definition within a key mapping list matches a key assignment with a command. The command can be defined in the application command table or it can be one of the DM-provided commands. The required KEY and CMD attributes of the KEYI tag match the key with the command.

The KEYI definition in this example matches the F2 key on the user's keyboard with the SEARCH command in the application command table.

```
<!doctype dm system>

<cmdtbl applid=abcd>
 <cmd name=search>Search
   <cmdact action=passthru>
</cmdtbl>

<keyl name=panlkeys>
  <keyi key=f2 cmd=search>
</keyl>

<panel name=panl01 keylist=panlkeys>
⋮
</panel>
```

When the user presses the F2 key during the display of the application panel that references this key mapping list, the Dialog Manager processes the SEARCH command.

Default Key Assignments

The Dialog Manager provides you with a set of default key assignments that are active during the display of any panel in your application, even if the panel does not name a key mapping list. These key assignments are:

Enter That runs the ENTER command.
Esc That runs the CANCEL command.
F1 That runs the HELP command.
F3 That runs the EXIT command.
F10 That runs the ACTIONS command.

You do not need to define any of these default keys in a key mapping list unless you want to override the Dialog Manager assignments for the keys. Overriding a default key assignment allows you to specify a different command for the key, or display the key assignment in the function key area of the panel by setting its FKA attribute to YES. For instance, you can override the F3 key assignment for EXIT by adding the F3 key to a key mapping list and assigning a different command to it.

```
<keyl name=panlkeys>
  <keyi key=f2 cmd=search>
  <keyi key=f3 cmd=down>
</keyl>
```

Combining Key Assignments

If you want to assign a combination of keys to an action (for example, Ctrl+F1) you can add the optional STATE attribute to a KEYI definition. STATE allows you to specify a shift key as part of a key assignment. The valid values you can assign to STATE are SHIFT, CONTROL, and ALT.

In this example, we add another key assignment to the list that specifies a combination of the Alt and F3 keys.

```
<keyl name=panlkeys>
  <keyi key=f2 cmd=search>
  <keyi key=f3 cmd=down>
  <keyi state=alt key=f3 cmd=delete>
</keyl>
```

Using the STATE attribute gives you a great deal of flexibility in defining key assignments. While Dialog Manager allows you to define many different combinations, you should be aware that there are some restrictions. Table 10-1 lists the keys and shift-states valid for Dialog Manager applications. Valid keys and combinations are indicated by a "Y."

Table 10-1 (Page 1 of 2). Valid Key and Shift State Combinations

Virtual-key	Alone	SHIFT	ALT	CONTROL
A–Z	.	.	Y	Y
BREAK	Y	.	.	.
ENTER	Y	Y	Y	Y
ESC	Y	.	.	.
F1–F24	Y	Y	Y	Y

Table 10-1 (Page 2 of 2). Valid Key and Shift State Combinations				
Virtual-key	**Alone**	**SHIFT**	**ALT**	**CONTROL**
PAUSE	Y	.	.	.
PRINTSCRN	Y	.	.	.

For example, a STATE value of SHIFT combined with F1 is allowable, as defined in this example:

```
<keyi key=f1 state=shift...>
```

However, this KEY assignment is not valid, because you cannot combine the Esc and Alt keys.

```
<keyi key=esc state=alt...>
```

Displaying Keys

While all of the key assignments you define in a key mapping list are valid for the application panels that reference the list, they will only appear in the function key area of the panel under the following conditions:

- You specify that the key can be displayed, and
- The user specifies the Dialog Manager FKA or CHGDEFS command.

You use the FKA attribute of the KEYI tag to specify whether the key is to appear in the panel's function key area. The default FKA value is NO, which means that the key will not appear unless you specify YES for the attribute.

When the F1 – F24 keys are displayed in the function key area, the key or key combination you assign is displayed followed by an equal sign that precedes the external-command name of the key assignment. When a key other than a function key is displayed, only the external command name appears.

If we wanted the F3 key to appear in the panel function key area, we would specify the FKA attribute like this:

```
<!doctype dm system>

<keyl name=panlkeys>
  <keyi key=f2 cmd=search>
  <keyi key=f3 cmd=down fka=yes>
  <keyi state=alt key=f3 cmd=delete>
</keyl>

<panel name=panl01 keylist=panlkeys>
⋮
</panel>
```

The function key looks like this in the function key area of the "panl01" application panel:

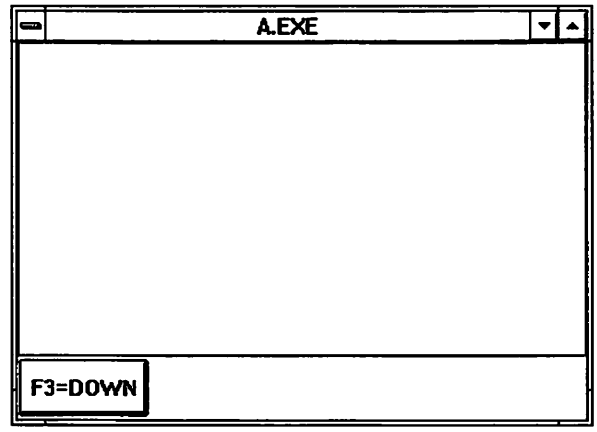


Figure 10-1. Displayed Function Key

Only those key assignments with an FKA value of YES appear in the function key area.

If you want the text of the function key to say something other than the *external-command-name* of the command it references, you can specify alternative text as tag content in the KEYI definition. For example, this KEYI definition defines the FKA text for the function key F2:

```
<!doctype dm system>

<keyl name=panlkeys>
  <keyi key=f2 cmd=search fka=yes>Locate
  <keyi key=f3 cmd=down fka=yes>
  <keyi state=alt key=f3 cmd=delete>
</keyl>

<panel name=panl01 keylist=panlkeys>
:
</panel>
```

Here is how the key appears in the function key area:

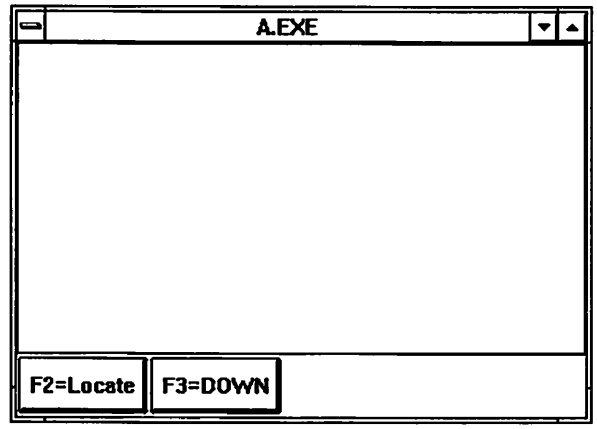


Figure 10-2. Displayed Function Key

When the Dialog Manager processes a key mapping list for display, it places the key assignments in the function key area in the following order, regardless of the order they are defined within the key mapping list:

1. Enter
2. Esc
3. F1...F24
4. Shift + F1...Shift + F24
5. Alt + F1...Alt + F24
6. Ctrl + F1...Ctrl + F24
7. Break
8. Pause
9. PrintScrn
10. Shift + Enter
11. Alt + A...Alt + Z
12. Alt + Enter
13. Ctrl + A...Ctrl + Z
14. Ctrl + Enter

If you want to display a default key assignment, you must define the key assignment in a key mapping list, and assign an FKA value of YES to the definition. This is the only method for displaying a default key.

In this example, we've defined a key mapping list that contains four key assignments that are defined for display in the function key area, and one key assignment that does not display. Because we want the key assignment for the Enter key to be displayed in the function key area, we defined a key assignment for it in the list and gave it an FKA value of YES.

```
<!doctype dm system>

<keyl name=panlkeys>
 <keyi key=f2 cmd=search fka=yes>Locate
 <keyi key=f3 cmd=down fka=yes>Backward
 <keyi state=alt key=f3 cmd=delete>
 <keyi key=enter cmd=enter fka=yes>
 <keyi state=control key=c cmd=copy fka=yes>Copy
</keyl>
⋮
<panel name=panl01 keylist=panlkeys>
⋮
</panel>
```

Here is the result. Notice how the Dialog Manager displays the key assignments according to the order described earlier, not the order they are defined in the key mapping list.

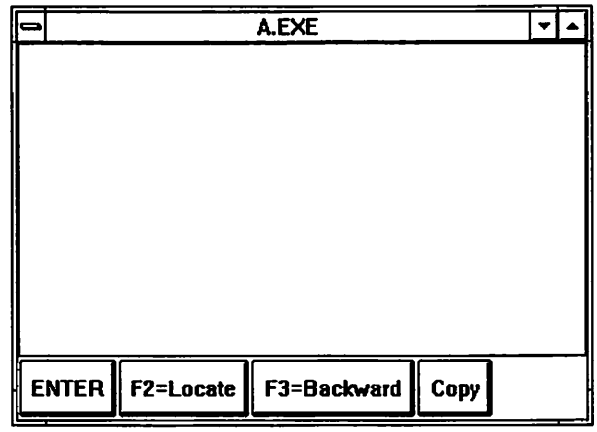


Figure 10-3. Function Key Area

Chapter 11. DTL Shortcuts

At this point, you probably have a good understanding of how the Dialog Tag Language works, and what is involved in marking up dialog elements and variables. And, while you've probably found that using DTL can save you a lot of time, there are more things you can do with DTL that will save you even more time—both when you create new source files, and when you revise existing ones.

This section shows you some of the ways you can use DTL to save steps in your markup.

Defining Panel Defaults

DTL provides you with a tag that makes it easier for you to define attributes and values that are common among multiple application panels: the PANDEF (panel default) tag. This tag must be coded in the source file before the panels it is providing defaults for.

The PANDEF tag allows you to define certain attributes and values that are common among multiple application panels in a single place, instead of repeatedly for each panel definition.

The default PANEL values you can define with the PANDEF tag are:

- The panel dimensions (DEPTH and WIDTH)
- The associated help panel
- The associated key mapping list.

You can use a PANDEF tag to define all of these values, or some of them. You can also override a specific panel default value for a referencing panel by specifying the attribute on the PANEL tag.

For instance, if you create a series of panels that all have the same dimensions and that all reference the same help panel and key mapping list, you can define these values in a PANDEF definition, and reference that definition in each of the application panels that use those values. The DTL compiler does the rest of the work for you, as long as you place the default definition in the same source file as the panels that reference it.

For example, if you are creating a series of panels that all share the same values, you could create a PANDEF definition like this:

```
<!doctype dm system>

<pandef id=printdef help=prnthelp depth=20 width=70 keylist=printkey>
```

And reference the panel default like this on all of the panels in that series:

```
<panel name=panel01 pandef=printdef>A Panel
⋮
</panel>

<panel name=panel02 pandef=printdef>Another Panel
⋮
</panel>
```

When you compile this source file, the PANDEF definition provides those values for the panels that reference the panel default.

You can also use the PANDEF tag to define common values for individual PANEL attributes. For instance, if the only commonality between application panels is the dimensions, you can use a panel default to define the dimensions and reference only those values from the application panels:

```
<!doctype dm system>

<pandef id=size depth=20 width=70>

<panel name=panel01 help=help01
keylist=shortkey pandef=size>A Panel
⋮
</panel>

<panel name=panel02 help=help02
keylist=longkey pandef=size>Another Panel
⋮
</panel>
```

Anytime you want to change the dimensions of the application panels that reference a panel default, you only have to make the change in one place—in the PANDEF definition.

To override a PANDEF value, you must specify that value in the PANEL definition. The following example contains a panel default that defines both dimensions and a help panel. While all three PANEL definitions reference the panel default, the panel with the NAME value *panel03* specifies a different help panel, and thus overrides the PANDEF HELP value.

```
<!doctype dm system>

<pandef id=pandef01 depth=20 width=70 help=help01>

<panel name=panel01 pandef=pandef01>
⋮
</panel>

<panel name=panel02 pandef=pandef01>
⋮
</panel>

<panel name=panel03 pandef=pandef01 help=help02>
⋮
</panel>
```

You can also define multiple panel defaults within a single source file, like this:

```
<!doctype dm system>
<pandef id=pandef01 depth=20 width=70 help=help01>

<pandef id=pandef02 depth=10 width=50 help=help02 keylist=klist01>

<panel name=panel01 pandef=pandef01>
⋮
</panel>

<panel name=panel02 pandef=pandef02>
⋮
</panel>

<panel name=panel03 pandef=pandef01 help=help02>
⋮
</panel>
```

Reducing Tag Markup

The Dialog Tag Language supports the SGML *minimization* feature, which allows you to eliminate some tags and reduce the overall complexity of your markup. For example, the use of optional, or *omitted* end tags is the most common way of minimizing markup, which is the style we've used in this book. However, there are many other ways you can use minimization to reduce tag markup.

DTL supports the following ways of reducing markup:

- Omitted optional end tags
- Empty start or end tags
- Unclosed start tags
- Unclosed end tags
- Null end tags.

Reducing markup can save you a great deal of time when marking up your source files. You should, however, have a proficient understanding of the Dialog Tag Language before attempting minimization, as unexpected results could occur.

Omitted Optional End Tags

This allows you to omit optional end tags. When you omit an optional end tag, the tag definition is implicitly ended by another tag that is not allowed within the scope of the tag.

The following examples illustrate how you can use omitted end tags:

Standard markup:

```
<note>This is a note explicitly ended by an
optional note end tag.
</note>
<p>This tag follows the note.
```

Reduced markup:

```
<note>This is a note implicitly ended by a
paragraph start tag.
<p>This tag implicitly ends the note.
```

Here's another example:

Standard markup:

```
<ol>
 <li>This is a list item explicitly ended
 with an optional end tag.
 </li>
 <li>Another list item, another optional end tag.
 </li>
</ol>
```

Reduced markup:

```
<ol>
 <li>This list item is implicitly ended by the next
 list item start tag.
 <li>This list item is implicitly ended by an ordered list
 end tag.
</ol>
```

Empty Start Tags

An empty start tag (< >) mimics the start tag that precedes it. This type of tag reduction is effective only when the preceding start tag has not been explicitly ended with an end tag.

Note: Do not use empty start tag reduction for tags with required attributes.

The following examples illustrate how you can use empty start tags:

Standard markup:

```
<ol compact>
  <li>This is the first list item in a compact ordered list.
  <li>This is the second list item.
  <li>This is the third list item.
</ol>
```

Reduced markup:

```
<ol compact>
  <li>This is the first list item in a compact ordered list.
  <>This is the second list item.
  <>This is the third list item.
</ol>
```

Empty End Tags

An empty end tag </> closes the most recent start tag. You can use empty end tags even when end tags are typically required.

Note: You cannot combine optional end tags and empty end tags. If you use empty end tags to end a succession of tags, you must explicitly end each start tag.

The following example illustrates how you can use empty end tags:

Standard markup:

```
<nt>
This is a note with a required end tag.
</nt>
```

Reduced markup:

```
<nt>
This is a note with a required end tag.
</>
```

You can also use empty end tag reduction to end a succession of start tags, as in this example:

Standard markup:

```
<panel name=infopan>An Information Panel
  <area>
    <info>
      <p>This is a paragraph within an information area
      that's within an area that's within a panel.
      Notice the end tags used to close the tags.
    </info>
  </area>
</panel>
```

Reduced markup:

```
<panel name=infopan>An Information Panel
  <area>
    <info>
      <p>This is a paragraph within an information area
      that's within an area that's within a panel.
      Notice the end tags used to close the tags.
      </>
    </>
  </>
</>
```

In the preceding example, every start tag was explicitly ended with an empty end tag.

Unclosed Start Tags

This type of reduction allows you to replace the close delimiter symbol (>) on a start tag with the open delimiter symbol (<) on the start tag that follows it.

Here's an example of an unclosed start tag:

Standard markup:

```
<ol>
  <li>This is a list item.
</ol>
```

Reduced markup:

```
<ol<li>This is a list item.
</ol>
```

Unclosed End Tags

This type of reduction allows you to replace the close delimiter symbol (>) on an end tag with the open delimiter symbol (</) on the end tag that follows it.

Here's an example of an unclosed end tag:

Standard markup:

```
<panel...>
  <area>
    <selfld...>
⋮
    </selfld>
  </area>
</panel>
```

Reduced markup:

```
<panel...>
  <area>
    <selfld...>
⋮
    </selfld</area</panel>
```

Null End Tags

This type of reduction allows null end tags (/) to be used. A null end tag closes the most recent start tag that is open and that is null-end-tag enabled. A start tag is null-end-tag enabled when it is closed with the slash right (/) symbol instead of a tag close delimiter symbol (>).

Standard markup:

```
<ol>
  <li>This is a list item.
  It is followed by an ordered list end tag,
  which ends the ordered list.
</ol>
```

Reduced markup:

```
<ol/
  <li>This is a list item.
  It is followed by a null end tag,
  which ends the ordered list.
/
```

Comprehensive Examples

You can also combine types of minimization in your markup. This example combines empty end tags and unclosed end tags.

Standard markup:

```
<panel...>
  <area>
    <dtafld...>
⋮
    </dtafld>
  </area>
</panel>
```

Reduced markup:

```
<panel...>
  <area>
    <dtafld...>
⋮
</</</>
```

This example combines empty start tags, unclosed start tags, and null end tags.

Standard markup:

```
<panel name=panel1>A Panel
  <area>
    <info>
      <p>This is a paragraph.
      <p>This is another paragraph.
      <ol>
        <li>First item in an ordered list.
        <li>Second item in an ordered list.
      </ol>
    </info>
  </area>
</panel>
```

Reduced markup:

```
<panel name=panel1>A Panel
<area<info>
<p>This is a paragraph.
<>This is another paragraph.
<ol/
<li>First item in an ordered list.
<>Second item in an ordered list.
/
</>
</>
</>
```

Chapter 12. Compiling and Displaying Dialog Elements

After you have finished coding dialog elements in your source files, you use the DTL compiler to create the elements and store them in DTL and HLP library files the Dialog Manager uses for your applications. HLP library files are used to store help panels that the OS/2 Information Presentation Facility (IPF) displays under the direction of the Dialog Manager. DTL library files are used to store other dialog elements managed by the Dialog Manager.

DTL provides you with commands to perform the steps that are necessary for preparing dialog elements for use in an application. These commands are:

DTLC For verifying tag syntax and creating formatted DTL and HLP library files.

DTLD For deleting and removing dialog elements from a DTL library. DTLD is not used for HLP libraries.

DTL also provides you with a display utility that allows you to view the formatted application panels and messages in your files, without an underlying DM program.

In this chapter, we show you how to use the DTL commands to prepare your files for use in DM applications. We also show you how to use the DISPLAY utility to list the formatted elements in a DTL library and see how panels and messages would appear when displayed in an application.

General Rules for DTL Commands

The following rules apply to all of the DTL commands:

- The DTL commands are issued on the OS/2 command line.
- The file names you specify with the DTL commands must follow standard OS/2 naming conventions. If the file name includes any of the special characters: semicolon (;), comma (,), plus (+) or the space character, then the file name must be enclosed in double quotes.
- If any of the DTL commands encounter a semicolon (;) in a command parameter, the semicolon is treated as the end of the parameter list, and all subsequent characters are ignored.
- All commands are positional and must be separated by commas.
- If you specify a list of names in a command, each name must be separated by either a blank or a plus sign (+).
- Equal signs (=) and colons (:) can be used interchangeably to separate an option and its value.
- All options associated with the DTL commands must be preceded by the slash (/) symbol.
- All options associated with the DTL commands can be abbreviated. For example, the /PRESERVE option associated with the DTLC command can be typed as /P.

 Note: Generally, you can abbreviate options to their first character. However, in the case of the /COMPRESS, /COUNTRY, and /CODEPAGE options,

the minimum abbreviations are the first three characters (/COM, /COU, and /COD, respectively).

Chapter 13, "DTL Command Reference" on page 13-1 provides a complete description of each of the DTL commands.

Compiling and Verifying Source Files

You use the DTLC command to compile the dialog elements in your source files. You can also use the DTLC command to verify the syntax in your source files. You perform the syntax check as a separate operation, or when you actually compile the source file.

Verifying Markup Syntax

If you want to verify the markup syntax in a source file and display the errors before you compile it, issue the DTLC command with the /SYNTAX option. When you specify the /SYNTAX option, the compiler does not create a file for the dialog elements in the source file.

For example, to check the syntax in a source file named MYTAGS.GML, issue the following command:

```
dtlc mytags /syntax
```

Using the minimum abbreviation of /SYNTAX, you could issue the same command this way:

```
dtlc mytags /s
```

This command produces the same result.

DTLC Messages

It is a good idea at this point to discuss compiler messages and exit values for the DTLC command. As we stated earlier, when the compiler processes a source file, it displays warning and error messages. Warning messages, in general, issue information about markup errors involving optional tag attributes. You should correct these errors and recompile the source file. Error messages indicate that an error in the markup is severe enough to prevent some of the associated dialog elements from being created. You must correct the error before attempting to format the file. In either case, the messages specify the line number and the cause of the error.

The compiler also returns an exit value when it completes the compile operation. Table 13-1 on page 13-7 shows the values and describes the meaning and result of each one.

In some cases, the compiler issues a message that references a line number in the source file. In these cases, one of two scenarios are possible:

- There is an error in the markup on that line in the source file
- There is an error in the markup before that line in the source file. In this instance, the error is not evident until that line is processed.

You can save compiler messages by directing them to a separate file. For example, to check the syntax in the MYTAGS.GML file and direct any resultant

warning and error messages to a file called OOPS.MSG, issue this DTLC command:

```
dtlc mytags /syntax>oops.msg
```

The > symbol preceding the specified message file indicates the destination of the messages issued by the compiler directed to the standard output device.

You can get additional help for a DTLC message by using the OS/2 HELP command.

Compiling Source Files

Use the DTLC command without the /SYNTAX option to compile a source file. For example,

```
dtlc mytags
```

In this case, the compiler not only checks the syntax of the source file and issues messages for errors and warnings, but it also formats the dialog elements within the source file and writes them to libraries named MYTAGS.DTL and MYTAGS.HLP.

If the compiler encounters an error, it stops writing the elements to the DTL library and does not create (or replace) the HLP library, although it continues to verify the syntax. If the compiler encounters a severe error, it halts all processing and exits. Table 13-1 on page 13-7 contains a list of the DTLC exit values.

Specifying a Library

Following the completion of the compile operation, the compiler produces DTL library files as its output. The compiler adds the extension .HLP to the file name to identify the file as a library containing help panels. The compiler adds the extension .DTL to the filename to identify the file as a library containing other dialog elements. For example, the library files created for the dialog elements in the source file MYTAGS.GML are named MYTAGS.DTL and MYTAGS.HLP.

The compiler creates a separate library file for the command table, and names the file by appending the string CMDS to the application identifier you specify for the command table. The compiler also adds the .DTL extension to the file name.

For example, if the application command table defined within the MYTAGS.GML file has the application identifier XYZ, the compiler creates the libraries MYTAGS.DTL, MYTAGS.HLP, and XYZCMDS.DTL.

If you want use a name other than the source file name for the output libraries, you can specify this with the LIBSPEC parameter of the DTLC command. For dialog elements stored in DTL libraries, you can place the elements defined by multiple source files within a single library by specifying the same LIBSPEC value when compiling each of the source files.

Note: HLP (help panel) library files are created or replaced in their entirety each time the DTL compiler is called. Because of this, you should be careful when specifying the LIBSPEC parameter that an existing HLP library is not overwritten.

For example, suppose the source file MYTAGS.GML contains the definitions for some of the application panels, messages, and key mapping lists for your application, and the source file MORETAGS.GML contains others. To compile and

store the elements in both of these source files within the same library named ALLTAGS.DTL, issue the following DTLC commands:

```
dtlc mytags,alltags
dtlc moretags ,alltags
```

You must separate the library specification from the source file name with a comma. In this example, we did this in two different ways. In the first DTLC call, we used only a comma. In the second DTLC call, we added a blank before the comma. We also could have added more blanks—it doesn't matter because the DTL compiler is only interested in the comma.

Specifying a Drive and Path

You can store library files in the current directory you are working in, in another directory on the same drive, or in another directory on a different drive.

For example, to direct the output library file for the MYTAGS.GML file to a directory named TAGS, issue the following command:

```
dtlc mytags, \tags\
```

To perform the same operation with a different library name, issue the command like this:

```
dtlc mytags, \tags\alltags
```

This directs the formatted elements to the ALLTAGS.DTL and ALLTAGS.HLP libraries located in the TAGS directory.

If the TAGS directory resides on a different drive (the D drive, for example), you would issue this command:

```
dtlc mytags, d:\tags\alltags
```

Replacing and Removing Existing Library Elements

If the output DTL library contains an existing element that has the same name as an element of the same type being compiled, the compiler marks the element as deleted from the library and adds the new element. To prevent this, you can use the /PRESERVE option of the DTLC command. This option specifies that the new element will not replace an existing element with the same name.

Similarly, the /PRESERVE option prevents a single HLP library from being overwritten in its entirety.

This DTLC command compiles and directs the output of the MYTAGS.GML file to the ALLTAGS.DTL and ALLTAGS.HLP libraries, and specifies the /PRESERVE option using the minimum abbreviation:

```
dtlc mytags, alltags /p
```

If there is an existing element in the ALLTAGS.DTL library that shares the same name as an element of the same type in the MYTAGS.GML file, the compiler issues a warning message and does not replace the existing element. If there is an existing ALLTAGS.HLP library, the compiler issues a warning message and the library will not be replaced.

When the compiler deletes an existing element within a DTL library, the element still occupies storage in the library, even though it cannot be accessed. To free

this unused storage, use the DTLC /COMPRESS option. We recommend that you use the /COMPRESS option frequently to conserve space in DTL libraries.

Note: You cannot use the /COMPRESS option for compressing HLP libraries.

This command directs the dialog elements in MYTAGS.GML to the ALLTAGS library and deletes and removes from the DTL library any previously deleted elements or any existing elements with identical name values.

```
dtlc mytags, alltags /com
```

Specifying System Information

Specifying a Country Code: The /COUNTRY (/COU) option allows you to specify a country-code for your library files. The default value for this option is the country-code currently in use on your operating system.

If you specify a country code other than the system country code, you must be sure that the country code is prepared in the system, and that the country code is identical to the country codes defined for all of the other elements in the DTL library. If you attempt to mix country codes within a specific DTL library, the compile operation will fail.

You can also specify a country ID in place of the country code. Table 13-2 on page 13-7 contains a complete list of the valid country IDs for the DTLC command, and the country codes they correspond to.

This DTLC command uses the minimum abbreviation for /COUNTRY to specify the country code for Denmark.

```
dtlc mytags, alltags /cou=045
```

This country code is used to prepare the MYTAGS.GML file. Additionally, all of the files within the destination library (ALLTAGS.DTL) must also have been prepared using the 045 country code.

We could have identified the same country code in the above example by specifying the valid country ID instead. The command would look like this:

```
dtlc mytags, alltags /cou=dk
```

Canada supports two country codes. If you use the country ID to identify the country-code, the compiler uses the first country code listed. For example, this command causes the compiler to use the country code "002" to prepare the file.

```
dtlc mytags , alltags /country=ca
```

Specifying a Code Page: The /CODEPAGE (/COD) option identifies the code page number used to prepare the file. The default value for /CODEPAGE is the code page number currently in use on your operating system.

If you specify a code page other than the system code page, you must be sure that the code page number is prepared in the system, and that the number is identical to the code page number defined for all of the other files in the DTL library. If you attempt to mix code page numbers within a specific DTL library the compile operation will fail.

For correct operation, the code page of the application's dialog variables should match the code page specified for the DTL source.

Note: OS/2 provides support for both ASCII and EBCDIC code pages. However, DTLC only supports ASCII code pages.

Specifying a Language: If you are preparing your application in a language other than U.S. English you need to specify a language ID for your source files when you compile them. You use the /LANGUAGE option of the DTLC command to specify the language ID. The minimum abbreviation for /LANGUAGE is /L.

This DTLC command call specifies the language ID for Danish:

```
dtlc mytags, alltags /cou=045 /l=dan
```

Table 13-3 on page 13-9 contains the valid language ID values, as well as the language and the name of the language file they correspond to.

Online Description

The DTL compiler provides you with an online description of the DTLC command and its associated options and parameters. To view the description, type the following command:

```
dtlc ?
```

Deleting Dialog Elements

You can use OS/2 commands to delete library files. If you want to selectively delete non-help dialog elements that are no longer needed and are not to be replaced by elements having the same name, you can use the DTLD command. Without the DTLD command, non-help dialog elements that have obsolete names would continue to occupy space within a DTL library.

When you specify dialog elements with the DTLD command, the elements are deleted from the library list, but still occupy storage space within the library unless you specify the /COMPRESS option.

You identify the dialog element to be deleted by the value assigned to the NAME attribute of the element, or, in the case of command tables, the application identifier appended to the string CMDS. Because different types of elements may share the same NAME value, you must also specify the type of element to be deleted. The types of elements you can specify for deletion are:

- CMDTBL (command table)
- ICON (icon resources)
- KEYL (key mapping lists)
- MSGMBR (message members)
- PANEL (application panels).

You can only specify one of the previous types of elements in each DTLD command call.

For example, to delete the application panel "ordpan01" from the MYTAGS.DTL library, issue the DTLD command like this:

```
dtld mytags, panel, ordpan01
```

DTLD assumes the library extension .DTL. The element type and element name are separated by commas in the command.

You can also specify multiple elements to be removed in a single DTLD command call. However, because you can only specify one element type in a DTLD command call, all of the elements must be of the same type. Each element must be separated by blanks or plus signs (+).

For example, this command removes the key mapping lists "key55," "key56" and "key88."

```
dtld mytags, keyl, key55 key56 key88
```

We could also issue this DTLD command like this:

```
dtld mytags, keyl, key55 + key56 + key88
```

This command removes a command table with the application identifier XYZ from the ABCCMDS.DTL library.

```
dtld abccmds ,cmdtbl ,xyzcmds
```

Compressing a DTL Library

As we stated earlier, when you delete an element from a library, the element still occupies storage space in the library, even though it cannot be accessed. Like the DTLC command, the DTLD command has a /COMPRESS option that you can use to remove a deleted element from the library, and free the unused storage space. In the following example, the DTLC command removes the message member "msga12" from the MYTAGS.DTL library, and compresses the storage space. The minimum abbreviation for /COMPRESS is used.

```
dtld mytags ,msgmbr ,msga12 /com
```

Specifying a Drive and Path

Like the DTLC command, you can specify alternate drives and paths when issuing a DTLD call. This DTLD command deletes the specified panels in the MYTAGS library, which resides in the TAGS directory.

```
dtld \tags\mytags ,panel ,panel1+panel2+panel3
```

Online Description

DTL provides you with an online description of the DTLD command and its associated options and parameters. To view the description, type the following command:

```
dtld ?
```

The DTL Display Utility

If you want to see what your formatted application panels and messages look like before you write the DM application code for them, you can use the display utility to display them. The application panels and messages you want to view must be in a .DTL file (a .GML file that has been compiled by the DTL compiler).

Note: You cannot use the display utility to view the help panels you create for your application.

Accessing the Display Utility

To use the display utility, enter the following command at the OS/2 command prompt:

```
DMDISPLY
```

Following the LOGO panel, the display utility main panel appears.

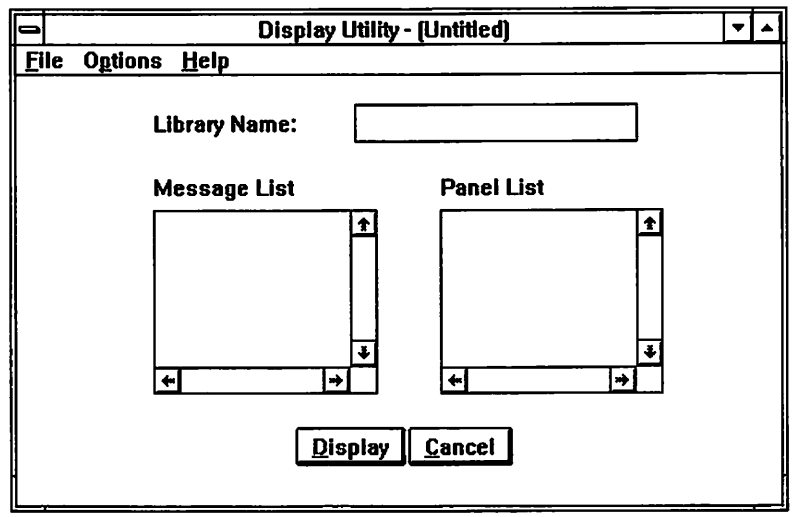


Figure 12-1. Display Utility Main Panel

Displaying Panels and Messages

There are two ways to access libraries from the main panel: with the **Library Name** field or the **File** action bar choice.

Library Name field

Enter the name of the library you want to access in the **Library Name** field. You must specify the full path, as well as the .DTL extension. You do not have to specify the path if the .DTL file has been added to your DPATH.

When you enter the library name, a list of the *message-identifiers* in the library is displayed in the **Message List** list box, and a list of the application panel NAME values in the library is displayed in the **Panel List** list box. Selecting a value from either one of these list boxes will display the formatted element.

File action bar choice

Select the **File** action bar choice to display its associated pull-down. Select the **Open** pull-down choice. This accesses the "Open File" panel, as shown in Figure 12-2 on page 12-9.

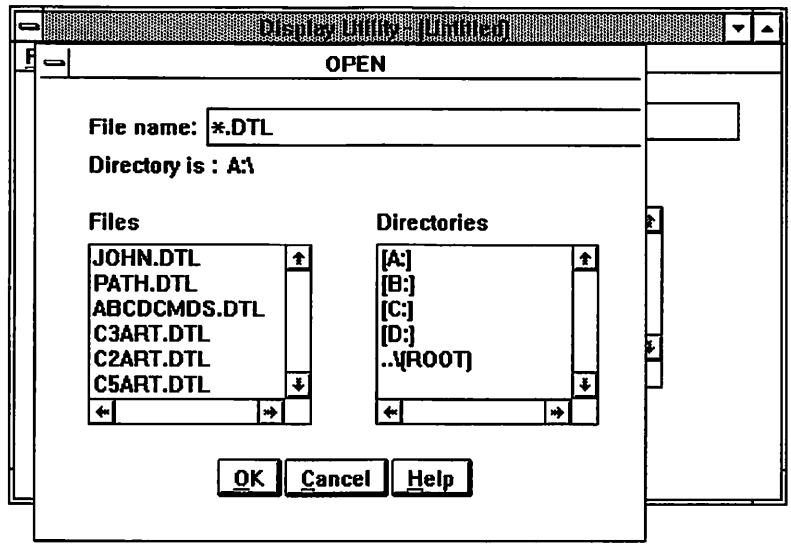


Figure 12-2. Display Utility Open File Panel

The current path name is displayed in the **Directory is:** field, and the default output "*.DTL" is displayed in the **File name:** field.

Two list boxes are also shown in the "Open" panel: **Files**, which displays a list of the libraries residing on the current path, and **Directories**, which displays a list of the current paths and directories. You can access the library that contains the elements you want to display from either list box.

- If the library that contains the element you want to view is in the current path, you can select the library name from the **Files** list box. The display utility main panel reappears showing a list of the *message-identifiers* in the library displayed in the **Message List** list box, and a list of the application panel NAME values in the library displayed in the **Panel List** list box. Selecting a value from either one of these list boxes will display the formatted element for the value.
- If the library that contains the element resides on a different directory, you can select the directory from the **Directories** list box. The **Directory is:** field is updated with the directory you selected, and the **Files** list box is updated with the list of libraries in the current path. Selecting a library name in the **Files** list box causes the display utility main panel to reappear. Selecting a value from either one of these list boxes will display the formatted element for the value.

Messages are displayed in a DM message panel. Application panels are displayed in windows.

You can exit from any displayed message or panel using the system icon **Close** pull-down choice.

Restrictions on the Display Utility

There are two types of panels that the display utility cannot process and therefore cannot display. These types include help panels and panels containing user controls.

Part 2. Dialog Tag Language Reference

About Part 2

This section contains the following chapters:

Chapter 13, "DTL Command Reference"
This chapter contains a reference listing for each the DTL commands.

Chapter 14, "Markup Declaration Reference"
This chapter contains a reference listing for each of the valid DTL markup declarations.

Chapter 15, "Tag Reference"
This chapter contains a reference listing for each of the DTL tags.

Each reference listing in this section contains a syntax diagram and attribute definition list, as well as a description and examples of usage.

How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

 The ▸▸— symbol indicates the beginning of a statement.

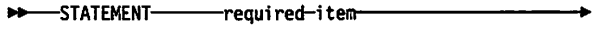
 The —▸ symbol indicates that the statement syntax is continued on the next line.

 The ▸— symbol indicates that a statement is continued from the previous line.

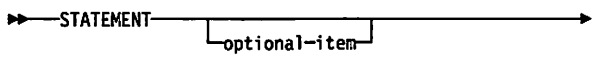
 The —▸◂ symbol indicates the end of a statement.

 Diagrams of syntactical units other than complete statements start with the ▸— symbol and end with the —▸ symbol.

- Required items appear on the horizontal line (the main path).

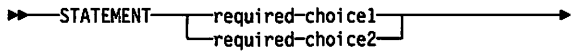


- Optional items appear below the main path.

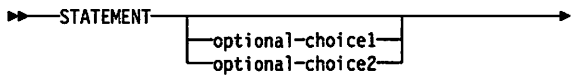


- If you can choose from two or more items, they appear vertically, in a stack.

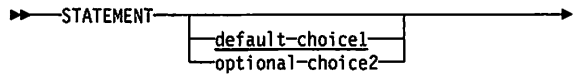
 If you *must* choose one of the items, one item of the stack appears on the main path.



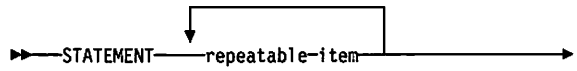
 If choosing one of the items is optional, the entire stack appears below the main path.



- Parameters that are underscored are default parameters. If you don't write it in the statement, you will get the same result as if you had actually written it.



- An arrow returning to the left above the item indicates an item that you can repeat. Required items appear on the main line and optional items appear below the main line.

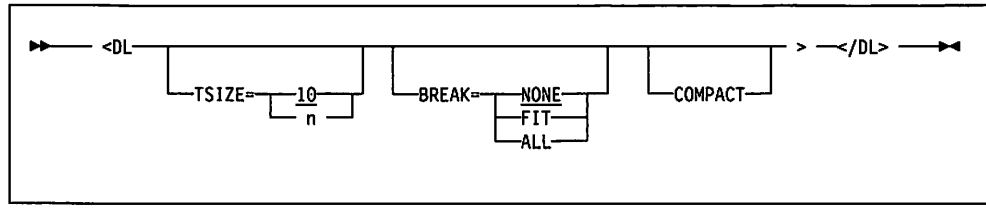


 A repeat arrow indicates that you can make more than one choice from the stacked items, or repeat a single item.

- Keywords appear in uppercase (for example, PARM1). However, they can be uppercase or lowercase when they are entered. They must be spelled exactly as shown. Variables and acceptable values appear in all lowercase letters (for example, parmx). They represent names or values that you supply. Keywords and keywords followed by keyword parameters (for example, MSG(message-id)) may be coded in any order.
- If punctuation marks, parentheses, arithmetic operators, or other symbols are shown, you must enter them as part of the syntax.

Syntax Example

The following diagram for the definition list (DL) illustrates the use of syntax diagrams.



Because DL appears on the main path, it is required and must be spelled exactly as shown; case does not matter.

Next are TSIZE, BREAK and COMPACT attributes, which are optional. Spell them exactly as shown; case does not matter. The TSIZE attribute has a default value of 10 or you can specify another value (*n*). Either NONE, FIT or ALL must be specified, but only if the BREAK attribute is coded. NONE is the default.

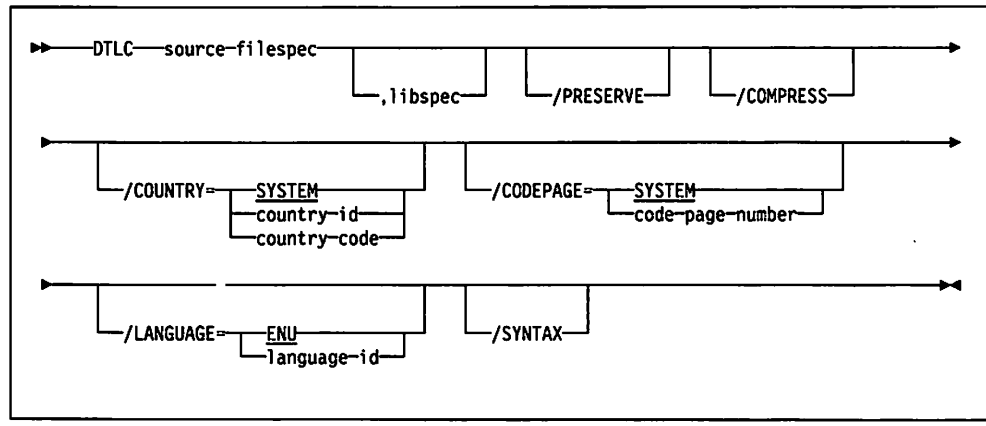
The DL end tag also appears on the main path, so it must be coded to end the definition list.

Chapter 13. DTL Command Reference

DTLC

The DTLC command creates dialog elements in HLP and DTL library files. HLP library files are used to store help panels that Interactive Presentation Facility displays under the direction of the Dialog Manager. DTL library files are used to store other dialog elements managed by the Dialog Manager.

The DTLC ? command displays a description of the DTLC parameters and options.



source-filespec

Identifies the source file to be compiled. If the file specification does not include a drive and path, the current directory is assumed.

The source-filespec can optionally include an extension. If you do not specify an extension, the compiler assumes the default extension .GML. Otherwise, the compiler will search for the file using the exact name given, and, if not found, searches again using a .GML extension appended to the source-filespec value.

The source file must contain DTL tags defining the dialog elements you are creating for your application. One source file can contain several or all of the dialog elements for an application.

Note: If the source-filespec includes any of the special characters: semicolon (;), comma (,), plus (+) or the space character then the file name must be enclosed in double quotes.

libspec

Identifies the DTL and HLP libraries that receive compiled application panels, help panels, messages, key mapping lists, and icons. It also identifies the directory in which the libraries are placed. Separate the *libspec* value from the source-filespec value with a comma. The compiler adds the extension .HLP to the name you specify for the *libspec* when creating or replacing the HLP library file containing help panels. The compiler adds the extension .DTL to the name you specify for the *libspec* when creating the DTL library file containing other dialog elements.

If you do not specify a *libspec*, the compiler constructs the library name using the source-filespec, and replaces its final extension with .HLP or .DTL, respectively. If the library file you specify does not currently exist, the library is created. Otherwise, the compiler adds or replaces the compiled elements in an existing DTL library and replaces the entire content of an existing HLP library.

The compiler places the libraries in the current directory unless you specify a different drive and path.

You do not need to specify a *libspec* if the source file contains only command table definitions. However, you can place a command table library in a specific directory by specifying a path without a library name.

Note: If the source-filespec includes any of the special characters: semicolon (;), comma (,), plus (+) or the space character then the file name must be enclosed in double quotes.

/PRESERVE

The minimum abbreviation is /P. For DTL libraries, /PRESERVE specifies that elements within the file specified in the source-filespec will not replace an existing element of the same type with the same name. If an element with the same name exists in the DTL library, the compiler issues a warning message. If you do not specify this option, any existing elements in the DTL library are replaced by new elements with the same name.

When the compiler replaces an element, it marks the existing element as deleted. A deleted element still occupies storage space in the library, unless you specify the /COMPRESS option to remove it from the library.

For HLP libraries, /PRESERVE specifies that if the HLP library already exists, it will not be overwritten.

/COMPRESS

Note: This option applies only to DTL libraries; HLP library files are replaced in their entirety.

The minimum abbreviation is /COM. Specifies that an existing element in a DTL library that is marked as deleted is also removed from the library. Elements are deleted from a DTL library when replaced by a new element with the DTLC command or deleted with the DTLD command. In these cases, the deleted elements still occupy the original amount of storage in the library. The /COMPRESS option removes the element and decreases the file size of the library.

/COUNTRY=SYSTEM | country-id | country-code

The minimum abbreviation is /COU. Identifies the country code used to process the translatable portion of the source file. The default is the system country code. The value of SYSTEM is defined by the configuration of your system.

You can also specify a *country-id* in place of a *country-code*. The valid *country-id* values and the country codes that they correspond to are shown in Table 13-2 on page 13-7.

All of the files in a specific library must have the same country code, whether they are specified or obtained through a default. Any attempt to mix country codes within a library will fail and result in an error message. In addition, there must be a prepared country code in the operating system for the specified *country-code*.

/CODEPAGE=SYSTEM | code-page-number

The minimum abbreviation is /COD. Identifies the code page used to process the translatable portion of the source file. The default is the system's current code page.

For correct operation, the code page of the application's dialog variables should match the code page specified for the DTL source.

All of the source files in a specific library must have the same code page, whether they are specified or obtained through a default. Any attempt to mix code pages within a library will fail and result in an error message. In addition, there must be a prepared code page in the operating system for the specified code page.

Note: OS/2 provides support for both ASCII and EBCDIC code pages. However, DTLC only supports ASCII code pages.

/LANGUAGE = ENU | language-id

The minimum abbreviation is /L. Identifies the language of the text in the specified source-file. The default value for this option is U.S. English. Table 13-3 on page 13-9 contains a list of the valid *language-id* values, as well as the language and the name of the language file the values correspond to.

The following list contains the keywords that must appear in language files, and their values in the ENU (U.S. English) file. The compiler adds these values to the formatted dialog elements. Use the UND *language-id* value and your own EDWNUND.NLS file to provide text translated in a language not supported by the compiler.

Note: Before compiling help panels, you should first copy the information in EDWNUND.NLS to the language file used by the OS/2 Information Presentation Facility compiler.

The language file used must reside in a directory specified by DPATH. Strings containing blanks must be enclosed within single or double quotes.

REM = text

Comment ignored by the DTL Compiler. A sample comment is shown in each of the language files. For instance, the English U.S. file, EDWNENU.NLS, contains the line

```
REM="U.S. English text"
```

This keyword can appear multiple times within a language file.

NOTE = "Note: "

Text embedded before the tag content of the NOTE and NT tags. This keyword can appear only once within a language file.

CAUTION = CAUTION:

Text embedded before the tag content of the CAUTION tag. This keyword can appear only once within a language file.

WARNING = "Warning: "

Text embedded before the tag content of the WARNING tag. This keyword can appear only once within a language file.

OLCHARS = abcdefghijklmnopqrstuvwxyz

Sequential characters used to precede the list items of an OL tag which is the second, fourth, sixth, eighth, or tenth ordered list in a nested group of OL tags. This keyword can appear only once within a language file.

OLCLOSE1 = .

Character(s) used to separate sequential numbers or characters from the list items of an OL tag which is the first, second, fifth, sixth, ninth, or tenth in a nested group of OL tags. This keyword can appear only once within a language file.

OLCLOSE2=)
Character(s) used to separate sequential numbers or characters from the list items of an OL tag which is the third, fourth, seventh, or eighth in a nested group of OL tags. This keyword can appear only once within a language file.

ULITEMID1=o
List item identifier for a UL tag which is the first, fourth, seventh, or tenth unordered list in a nested group of UL tags. This keyword can appear only once within a language file.

ULITEMID2=-
List item identifier for a UL tag which is the second, fifth, or eighth unordered list in a nested group of UL tags. This keyword can appear only once within a language file.

ULITEMID3=--
List item identifier for a UL tag which is the third, sixth, or ninth unordered list in a nested group of UL tags. This keyword can appear only once within a language file.

/SYNTAX
The minimum abbreviation is /S. Indicates the source-filespec should be syntax checked only. No library files are created if you specify this option, even if you provide a library specification.

Description

The DTLC command creates dialog elements in HLP or DTL library files. HLP library files are used to store help panels that the OS/2 Information Presentation Facility displays under the direction of the Dialog Manager. DTL library files are used to store other dialog elements managed by the Dialog Manager.

When used to create help panels, DTLC converts your source to a form accepted by the IPFC command of the OS/2 Information Presentation Facility, writes that source to a temporary file, and calls IPFC to create the HLP (help panel) library file. After the temporary file has been processed by IPFC, this file is deleted by DTLC.

During compilation, dialog elements (other than help panels) are written to the designated DTL library file while converted help panel source is written to a temporary file. Additionally, the compiler places the application command table in a separate library file. The name of this file is composed of the specified application-identifier for the command table with the string CMDS appended to it. The compiler adds the extension .DTL.

Next, IPFC is called to compile the temporary file and create or replace the HLP library file.

Names of libraries should follow the OS/2 file naming conventions.

The options associated with the DTLC command allow you to specify processing preferences and any special characteristics of the source file.

If you want to verify the syntax of a source file without performing the compile operation, specify the /SYNTAX option with the DTLC command. If you do not specify /SYNTAX, the compiler will still check the syntax for errors, in addition to performing the compile operation.

In either case, when the compiler checks the source file, it tries to find as many errors as possible. It accomplishes this by "fixing" the error in the source and continuing with the compile operation. Most of the time, this allows all errors to be identified. Sometimes, the "fix" applied by the compiler creates new errors or produces a state where further syntax checking is stopped. If this situation occurs, correct the syntax in the source file and recompile.

If errors or severe errors are detected, some dialog elements may have been added or replaced within a DTL library file before the errors were detected. If errors are detected by DTLC, IPFC is not called to create or replace a HLP library file.

Any errors reported by IPFC reference the original file name and line number of the DTL source. At times, the message may appear to reference a tag that does not specifically appear in the DTL source. This can occur because the DTL compiler converts DTL tags to a form acceptable to IPFC.

The following list shows some relationships that may be useful:

IPF Tag	DTL Tag
H1	HELP
I1	ITOP
LINK	RP

Example

This DTLC command checks the syntax of the markup in the ABCTAGS.GML file and performs the compile operation. The formatted elements are stored in libraries called ABCTAGS.DTL and ABCTAGS.HLP (the default names).

```
dtlc abctags
```

This DTLC command checks the syntax of the markup in the ABCTAGS.GML file and stores any error and warning messages directed to the standard output device in the ABC.MSG file. No dialog elements are created.

```
dtlc abctags /s >abc.msg
```

This DTLC command checks the syntax of the markup and stores the dialog elements defined by the ABCTAGS source file in libraries called XYZTAGS.DTL and XYZTAGS.HLP residing in the TAGS directory.

```
dtlc abctags, \tags\xyztags
```

Output

Compiler copyright notices, terminating messages, and completion messages (indicating the number of messages issued) are written to the standard error device. All other messages are written to the standard output device.

Exit Values

Table 13-1 on page 13-7 shows the exit values for DTLC, and describes the meaning and result of each one.

Table 13-1. DTLC exit values		
Exit value	**Meaning**	**Result**
0	The compiler found no markup errors, and issues no warnings or errors.	The dialog elements are created, unless the /SYNTAX option was specified.
1	The compiler issues a warning message.	The dialog elements are created, unless the /SYNTAX option was specified, regardless of the number of warnings are generated.
2	The compiler issues an error message.	No additional dialog elements are created, although syntax checking continues.
4	The compiler issues a severe error message.	The compiler ends all processing.

Option Values

Table 13-2 contains a list of the valid country IDs and their corresponding countries and *country-codes* for the /COUNTRY option.

Table 13-2 (Page 1 of 2). Country ID list		
ID	**Country**	**Country-code**
AR	Argentina	003
AU	Australia	061
BE	Belgium	032
BO	Bolivia	003
BR	Brazil	003
CA	Canada	002, 001
CH	Switzerland	041
CL	Chile	003
CN	People's Republic of China	086
CO	Colombia	003
CR	Costa Rica	003
DD	German Democratic Republic	049
DE	Federal Republic of Germany	049
DK	Denmark	045
EC	Ecuador	003
ES	Spain	034
FI	Finland	358
FR	France	033

Table 13-2 (Page 2 of 2). Country ID list		
ID	**Country**	**Country-code**
GB	United Kingdom	044
GT	Guatemala	003
GY	Guyana	003
HK	Hong Kong	099
HN	Honduras	003
IT	Italy	039
JP	Japan	081
KR	Republic of Korea	082
MX	Mexico	003
NI	Nicaragua	003
NL	Netherlands	031
NO	Norway	047
PA	Panama	003
PE	Peru	003
PT	Portugal	351
PY	Paraguay	003
SE	Sweden	046
SR	Suriname	003
SV	El Salvador	003
TW	Republic of China	088
US	United States	001
UY	Uruguay	003
VE	Venezuela	003

Table 13-3 contains a list of the valid language IDs for the /LANGUAGE option.

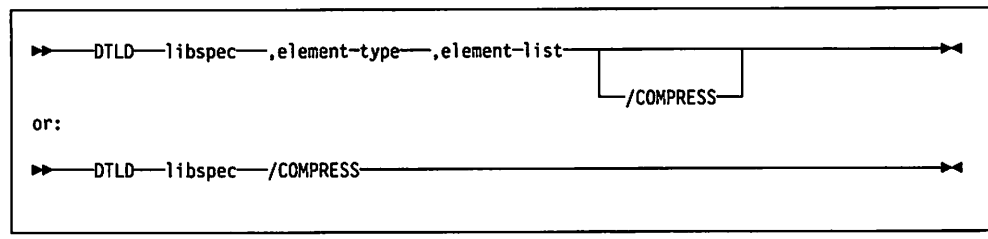
Table 13-3. Language ID list

ID	Language	Language file
DAN	Danish	EDWNDAN.NLS
DEU	German	EDWNDEU.NLS
ENG	English UK	EDWNENG.NLS
ENU	English US	EDWNENU.NLS
ESP	Spanish	EDWNESP.NLS
FIN	Finnish	EDWNFIN.NLS
FRA	French	EDWNFRA.NLS
FRC	Canadian French	EDWNFRC.NLS
ITA	Italian	EDWNITA.NLS
NLD	Dutch	EDWNNLD.NLS
NOR	Norwegian	EDWNNOR.NLS
PTG	Portuguese	EDWNPTG.NLS
SVE	Swedish	EDWNSVE.NLS
UND	Undefined (user-defined)	EDWNUND.NLS

DTLD

The DTLD command removes dialog elements from DTL libraries.

The DTLD ? command displays a description of the DTLD parameters and options.



libspec
Identifies the DTL library from which the named dialog elements are to be removed.

DTLD searches the current directory for the specified library unless you specify a different drive and path. If you do not provide an extension, DTLD uses the default .DTL extension. Otherwise, DTLD will search for the file using the exact name given, and, if not found, searches again using a .DTL extension appended to the *libspec* value.

Note: If the libspec includes any of the special characters: semicolon (;), comma (,), plus (+) or the space character, then the libspec must be enclosed in double quotes.

element-type
Specifies the type of element to be deleted. Separate the *element-type* value from the *libspec* value with a comma.

The value you can assign as an *element-type* must one of the following:

- CMDTBL
- ICON
- KEYL
- MSGMBR
- PANEL.

You can specify only one of the above values as an *element-type* for each DTLD command call.

element-list
Specifies the names of the dialog elements to be removed from the library. Separate the *element-list* value from the *element-type* value with a comma. In addition, each name in the *element-list* must be separated by blanks or plus signs (+).

If DTLD does not find a name in the *element-list*, it issues a message and continues searching for the remaining elements.

Because you can specify only one *element-type* in each DTLD command call, the elements you specify in the *element-list* must of the same type.

/COMPRESS
The minimum abbreviation is /COM. Specifies that the elements named in the *element-list* are removed in addition to being deleted. Elements are deleted from a library when replaced by a new element with the DTLC command or deleted with the DTLD command and the /COMPRESS option is not specified.

In these cases, the deleted elements still occupy the original amount of storage in the library. The /COMPRESS option removes the element and decreases the file size of the library.

To use this option, it must be possible to create a temporary file equal in size to the library being compressed.

The second form of the command compresses the library to remove any deleted elements. No other elements are deleted, because an *element-list* is not specified.

Description

The DTLD command removes one or more dialog elements from a single library. The values you specify for the *element-list* are the NAME values of the elements to be deleted. If the *element-type* is CMDTBL, then the value you assign in the *element-list* must be the application identifier for the command table appended to the string CMDS.

You can specify one or more element names to be deleted. However, all of the elements named in a single DTLD command must of the same type.

Use the /COMPRESS option to remove deleted elements from storage.

Example

The following DTLD command removes the application panels "pan01," "pan02," and "pan45" from the INVOICE library.

```
dtld invoice ,panel ,pan01 pan02 pan45
```

This DTLD command removes the key mapping list elements "key34," "key35," and "key36" from the INVKEY library. The /COMPRESS option specifies that the deleted elements are removed from storage in the library.

```
dtld invkey ,keyl ,key34+key35+key36 /COMPRESS
```

Output

The copyright notice and severe error messages (those messages causing DTLD to end abnormally) are directed to the standard error device. All other messages are written to the standard output device.

Exit Values

Table 13-4 shows the exit values for DTLD, and describes the meaning and result of each one.

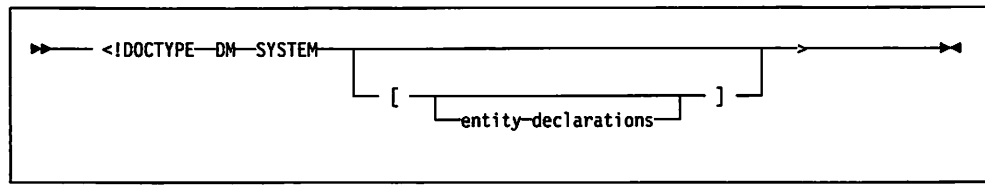
Table 13-4. DTLD exit values

Exit value	Meaning	Result
0	DTLD found no errors, and issues no warnings or errors.	The DTLD call completes successfully.
4	DTLD issues a severe error message.	All processing ends.

Chapter 14. Markup Declaration Reference

Document Type Declaration

The document-type declaration (DOCTYPE) identifies the source file document type and the rules the source file must follow.



DOCTYPE
Indicates that this is a document-type declaration.

DM
Indicates that this is a DTL source file defining dialog elements.

SYSTEM
Indicates that the rules for the DOCTYPE are contained in an external file.

[
Indicates the beginning of the declaration subset. The declaration subset may contain entity declarations and parameter entity references.

entity-declarations
The entity declarations you define for the source file must be coded within the declaration subset. "Entity Declarations" on page 14-3 contains a complete description of entity declarations.

]
Indicates the end of the declaration subset.

Description

A document-type declaration identifies the source file document type and rules the source file must follow.

The DOCTYPE declaration must appear in a DTL source file before any tag markup, although it may be preceded by comments. Files that are embedded in a source file intended for compilation cannot contain a DOCTYPE declaration.

Example

The DOCTYPE statement declares the source file as a DM type file.

```
<!doctype dm system>

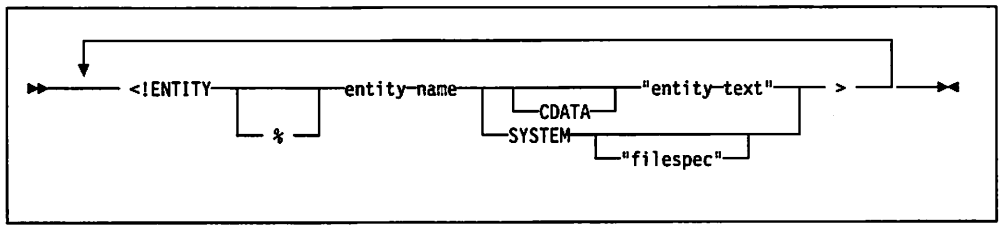
<varclass name=varc type='char 10'>

<varlist>
  <vardcl name=vard varclass=varc>
</varlist>

<panel name=panel>
:
```

Entity Declarations

Entities are symbolic names that are used to insert text into a file.



ENTITY
Indicates this is an entity declaration.

%
Indicates a parameter entity declaration. Must be followed by at least one space.

entity-name
The name of the entity.

It must follow these rules:

- One to eight (seven, if it is a parameter entity) characters in length.
- The first character must be alphabetic (a-z or A-Z).
- Remaining characters, if any, must be A-Z, a-z, 0-9, period (.), or hyphen (-).
- Entity-names are case-sensitive.

CDATA
Indicates that any delimiter characters in *entity-text* will not be interpreted as delimiters. This allows you to define entities with tags in *entity-text* that will not be interpreted as tags.

For example the entity-text " <panel> " will not be interpreted as the PANEL tag if the CDATA keyword is used.

Note: CDATA cannot be used with parameter entities.

"entity-text"
The text associated with the entity reference. The text must be enclosed in single or double quotes. This text must be less than or equal to 253 bytes.

SYSTEM
Indicates this entity refers to an external file.

filespec
The name of the file the entity refers to. If this is not specified, it defaults to the name of the entity with an extension appended. The extension depends on the type of entity.

.GML is used for entities.
.SPE is used for parameter entities.

Description

Entities are symbolic names that are used to insert text into a file. The text that an entity refers to can be a simple string of characters or it can be the text from an entire file.

An entity reference is used to insert the text associated with the entity. Entities must be declared in the declaration subset of the DOCTYPE declaration before they can be referenced. To reference the entity in the source file, the entity name is preceded by an ampersand (&) to indicate it is an entity or percent (%) to indicate it is a parameter entity. Both types of entities are ended with a semicolon (;). A blank or the end of the line may be used to end the entity reference instead of the semicolon.

References to entities can be made anywhere in the source file *after* the end of the DOCTYPE declaration.

References to parameter entities can only be made in the declaration subset after they have been declared.

Since entity declarations can only be made within the declaration subset, the parameter entity is the only way to embed a file of entity declarations. Parameter entities are used when an entity reference is needed in the declaration subset.

Conditions

If an end of line ends an entity reference, the end of line character is no longer processed as an end of line. Use a semicolon to end the entity reference if the end of line should be preserved.

An entity reference to an entity that has been declared is not required.

If a SYSTEM entity file cannot be found in the current directory, the DPATH is searched for the file. A full path may be specified for the *filespec* parameter if the file is not in the current directory or located on the DPATH.

Example

This example uses both entities and parameter entities. It embeds the file GLBENT.SPE with global entity declarations, and a file with tags and text. It also uses entities and parameter entities that refer to text strings.

The first entity declaration declares the "glbent" parameter entity as an external file.

The file name is defaulted to GLBENT.SPE. A parameter entity is used because this file contains entity declarations. Because entity declarations can only be made in the declaration subset, the GLBENT.SPE. file is embedded with an entity reference within the declaration subset. The entity declarations in GLBENT.SPE. are for text that is used at the top and bottom of the panel. The "header" entity declaration refers to an external file, and the "footer" is a text string. Both of these entities are referenced in the source file.

The second entity declaration, for "list," is also a parameter entity. This declaration refers to a string, not an external file. The text is the SL tag name, which is referenced in the next two entity declarations. These two declarations, "slist" and "elist," are used as the SL start and end tags. They are defined as entities so the type of list can be changed in one place. To change the list type

from a simple list (SL) to an unordered list (UL), change the parameter entity "list" from SL to UL.

This is the content of the source file:

```
<!DOCTYPE DM SYSTEM [
<!ENTITY % glbent SYSTEM -- declaration of global entity file -->
%glbent;  <!-- Embed global entity file -->
<!ENTITY % list "sl" -- type of list -->
<!ENTITY slist "<%list;>" -- type of list start tag. -->
<!ENTITY elist "</%list;>" -- type of list end tag. -->
]>

<panel name=showlist depth=22 width=45>Show Departments
  <area>
    <info width=40>
      &header;
      <p>The floors and departments are shown below:
      &slist;
        <li>First floor
          &slist;
            <li>Toys
            <li>Electronics
          &elist;
        <li>Second floor
          &slist;
            <li>Boys clothes
            <li>Girls clothes
          &elist;
      &elist;
      &footer;
    </info>
  </area>
</panel>
```

This is the content of the embedded file GLBENT.SPE:

```
<!ENTITY header SYSTEM "coname.gml">
<!ENTITY footer "<p>We're always glad to help!">
```

This is the content of the embedded file CONAME.GML:

```
<lines>
Jeff's Children's World
Barnett NC
</lines>
```

Figure 14-1 shows the formatted result:

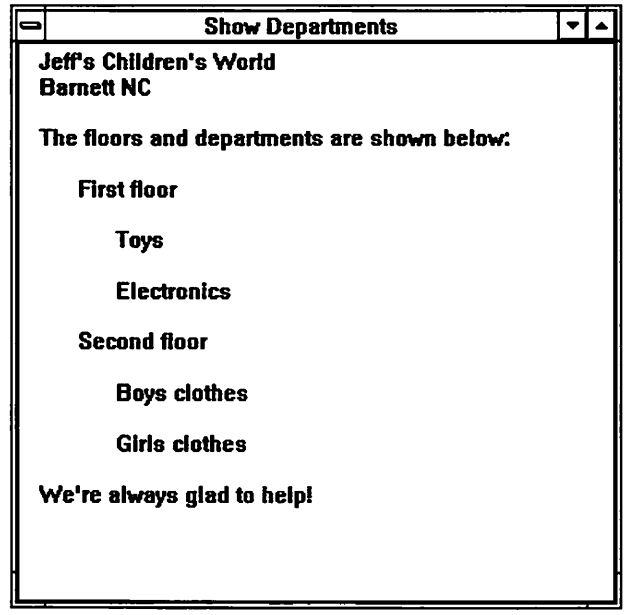


Figure 14-1. Entities and Parameter Entities

Chapter 15. Tag Reference

This chapter contains an alphabetical reference of the Dialog Tag Language (DTL).

This reference contains a description of syntax conventions, attribute descriptions, functional descriptions, conditions, tag nesting information, and examples for the tags.

Each tag listing includes a syntax diagram that illustrates the attributes available for that tag. For a description of how to use these diagrams, see "How to Read the Syntax Diagrams" on page 12-11.

How to Use the Nested Tag Tables

Throughout the tag reference section, the nesting characteristics of tags are described using the nested tag tables.

Note: These tables show only the primary nesting characteristics of tags, that is, only the tags you can nest directly within a specific tag are listed.

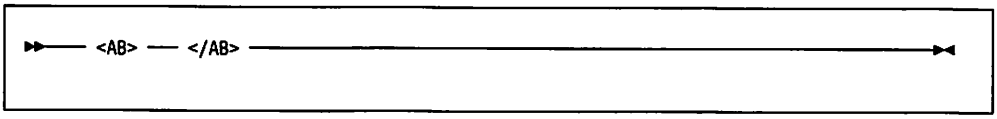
The characteristics are listed as follows:

- The tag name and description are listed in the **Tag** column of the nested tag table.
- The **Dialog Element** column describes whether you can code a tag within a specific element such as an application panel, a help panel, or a message member.
- The **Usage** column lists whether you can code a nested tag multiple times within a specific tag.
- The **Page** column lists the page where the description of the nested tag is located.

Note: In some cases, the tag nesting rules supported for help tags are more restrictive than the rules for the same tags coded within application panels. The **Dialog Element** column of the Nested Tags section of each tag description specifies which tags can be nested for application panels and for help panels.

AB (Action Bar)

The AB tag defines an action bar on an application panel.



Description

The AB tag defines an action bar on an application panel. The action bar will appear on the panel below the window title. The action bar provides a way to view all actions that apply to the panel that will be displayed. Action bar choices are keywords that appear on the action bar. They are used to group related actions. A list of actions, or pull-down choices related to an action bar choice, appear in a pull-down menu when an action bar choice is selected.

To define an action bar, you code the AB tag within the PANEL definition. The action bar choice is defined with the ABC tag and the pull-down choice or choices are defined with the PDC tag. The ABC tag is coded within the AB tag. The PDC tag is coded within the ABC tag.

Conditions

A panel definition can include only one action bar.

If your application panel contains an action bar, it must contain a **Help** action bar choice to conform to CUA.

The AB tag must be coded immediately after the PANEL tag.

Nested Tags

You can code the following tag within the AB tag:

Tag	Dialog Element	Usage	Page
ABC (action bar choice)	Application Panel	Multiple	15-5

Example

The following markup shows the tag source for a common user access (CUA) conforming action bar. The tag source for the **File** pull-down is also shown. Figure 15-1 on page 15-4 shows the formatted result for the action bar and the **File** pull-down.

```
<!doctype dm system>

<panel name=ab1 width= 40 depth=10>Catalog Ordering System
  <ab>
    <abc><m>File
      <pdc><m>New
        <action run=New>
      <pdc><m>Open...
        <action run=Open>
      <pdsep>
      <pdc><m>Save
        <action run=Save>
      <pdc>Save <M>as...
        <action run=SaveAs>
      <pdsep>
      <pdc>E<m>xit
        <action run=Exit>
    </abc>

    <abc><m>View

    <abc><m>Options

    <abc><m>Help

  </ab>
  <info>
  </info>
</panel>
```

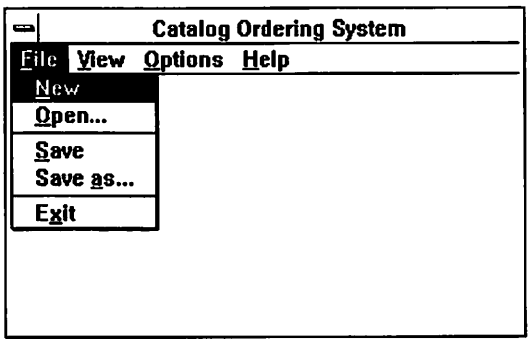
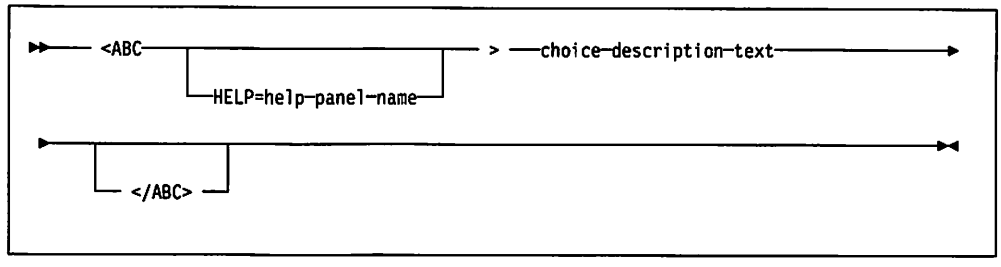


Figure 15-1. Action Bar and Pull-Down.

ABC (Action Bar Choice)

The ABC tag defines a keyword or choice in an action bar, and serves as a base for the pull-down choice tags.



HELP = help-panel-name
This attribute names a help panel that displays when the user selects help on the choice. See "HELP (Help Panel)" on page 15-75 for information on creating help panels.

If the user selects help on an action bar choice and no help exists, the extended help panel is displayed. If an extended help panel was not defined for this panel, a message is displayed stating that no help is defined. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

choice-description-text
This specifies the text that appears in the action bar. The text is limited to 20 bytes. You can use an M tag to identify a mnemonic character for the action bar choice.

The use of mnemonics must be consistent within an action bar. That is, each action bar choice must have a unique mnemonic or no mnemonics should be defined. The mnemonic character is underscored at run time.

Description

The ABC tag defines a keyword or choice in an action bar, and serves as a base for the pull-down choice tags. Actions or pull-down choices appear in a pull-down menu when the action bar choice is selected.

Conditions

You must code the ABC tag within an AB tag.

Nested Tags

You can code the following tags within the ABC tag:

Tag	Dialog Element	Usage	Page
M (mnemonic)	Application Panel	Single	15-110
PDC (pull-down choice)	Application Panel	Multiple	15-142
PDSEP (pull-down separator)	Application Panel	Multiple	15-145

Example

The following markup is a subset of the markup shown for the AB tag. (See "AB (Action Bar)" on page 15-3 for more information.) In addition to what is shown in the AB tag markup, help panels can also be added to the ABC. The following markup shown for the ABC tag shows only the action bar choices, and specifies a help panel for each action bar choice. Dialog Manager supplies a default help panel that you can use for the Dialog Manager supplied **Help** action bar choice. The name of the **Help** action bar choice help panel is *ispnabc*.

Notice that mnemonics <M> can be specified on action bar choices so that the keyboard user has a fast path for selecting an action bar choice.

```
<!doctype dm system>

<panel name=abc1>Catalog Ordering System
  <ab>
    <abc help=hlpfile><m>File

    <abc help=hlpview><m>View

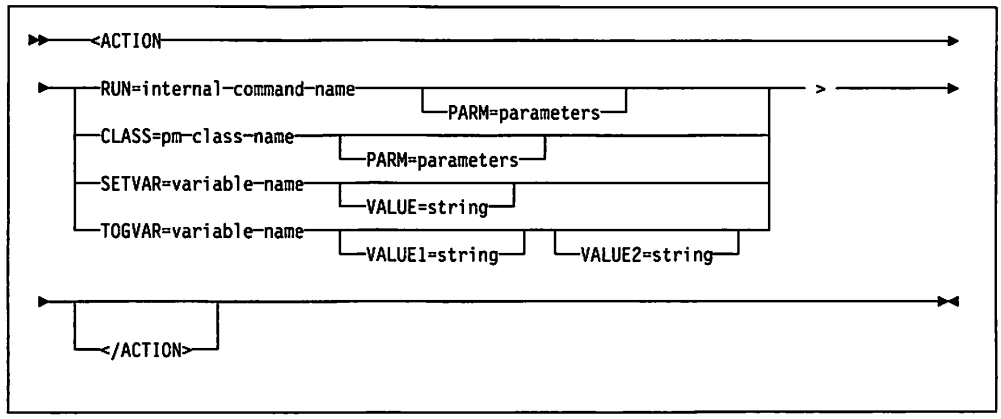
    <abc help=hlpopt><m>Options

    <abc help=ispnabc><m>Help
  </ab>

  <info>
  </info>
</panel>
```

ACTION (Action)

The ACTION tag defines the action that occurs when a pull-down choice or a selection field choice is selected.



RUN = internal-command-name

This attribute specifies the name of the internal command to run. The search for the command follows the normal command processing rules.

Note: The RUN action can be used only when the ACTION tag is coded under the PDC tag.

The RUN action is an ending action. Thus, if multiple ACTION tags are coded for a given pull-down or selection field choice, those following a RUN action are ignored. The *internal-command-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

The *internal-command-name* is used in two ways:

- As the command table search criteria when a pull-down choice or selection field choice with an associated RUN action is selected.
- As the value passed to applications when the command action is PASSTHRU or SETVERB. See "CMDACT (Command Action)" on page 15-37 for more information on the PASSTHRU and SETVERB command actions.

PARM = parameters

This attribute specifies the command parameters that are passed to command processing with the command referenced on the RUN attribute. Command processing handles the parameters entered in the command area and parameters entered on the PARM attribute in a similar manner. You can specify the name of a dialog variable (using the % notation) whose value at run time will be passed as the parameter data.

CLASS = pm-class-name

The CLASS attribute specifies the class name of the action user exit which is supplied by the application. For more information, refer to the user exits section of the *Dialog Manager Guide and Reference*.

PARM = parameters
Specifies parameters for the action user exit. You can specify the name of a single variable (using the % notation) whose value at run time will be passed as the parameter data.

SETVAR = variable-name
The SETVAR attribute names the variable whose value is updated as a result of selecting the associated choice. The attribute value must be coded without % notation.

VALUE = string
This attribute sets the value of the variable named on the SETVAR attribute. If you code the SETVAR attribute but omit the VALUE attribute, the Dialog Manager assigns the variable a value of 1. You can specify the name of a single variable (using the % notation) whose value at run time will be used to set the variable.

TOGVAR = variable-name
This attribute allows you to alternate the value of a single variable between two values. The TOGVAR attribute names the variable to set. The attribute value must be coded without the % notation.

The function of the TOGVAR action can be shown as follows:

```
if (TOGVAR-variable-name = VALUE1-string)
  TOGVAR-variable-name = VALUE2-string
else
  TOGVAR-variable-name = VALUE1-string
```

VALUE1 = string
This attribute sets the value of the variable named on the TOGVAR attribute if it is not currently equal to this value. VALUE1 defaults to 0 if not specified. You can specify the name of a single variable (using the % notation) whose value at run time will be used to set the variable.

VALUE2 = string
This attribute sets the value of the variable named on the TOGVAR attribute if it is currently equal to the value specified with the VALUE1 attribute. VALUE2 defaults to 1 if not specified. You can specify the name of a single variable (using the % notation) whose value at run time will be used to set the variable.

Description

The ACTION tag defines an action that occurs when a pull-down choice or a selection field choice is selected. You can specify multiple ACTION tags for a given choice; however, the ending action, RUN, will cause all ACTION tags that follow it to be ignored at run time. The Dialog Manager carries out the actions in the order in which you code them.

The ACTION tag and its RUN attribute should be coded following all pull-down choices and pushbuttons they are associated with. Pull-down choices and pushbuttons indicate an immediate action to the user and coding an action with the RUN attribute ensures that an immediate action will occur when the user selects the item.

Single- and multiple-choice selection fields do not indicate an immediate action to the user, so you should not use the RUN attribute on an ACTION tag that is under these types of selection field choices. Typically, the ACTION tag coded under a single- or multiple- choice selection field is used to set or toggle a variable when

that choice is selected. The variables set or toggled can be used on other fields in the panel, so that selecting the choice would cause another field on the panel to immediately be de-emphasized or selected without any involvement from your application code.

On the PARM, VALUE, VALUE1 and VALUE2 attributes of the ACTION tag, you must code an additional % before any string beginning with a %, to distinguish it from a variable name. (For example, to specify the string %abc, code '%%abc'; to specify the variable abc, code '%abc'.)

Conditions

You can code this tag only within a CHOICE or a PDC tag.

Nested Tags

None.

Example

Pull-Down Choice Actions: In the following markup, pull-down choices indicate immediate actions to the user. This example shows how the ACTION tag is used to define actions that occur when a pull-down choice is selected. The actions for the **File** and **Help** pull-downs are shown here.

```
<!doctype dm system>
<panel name=action1 width=40>Catalog Ordering System
  <ab>
    <abc><m>File
      <pdc><m>New
        <action run=New>
      <pdc><m>Open...
        <action run=Open>
      <pdsep>
      <pdc><m>Save
        <action run=Save>
      <pdc>Save <m>as...
        <action run=SaveAs>
      <pdsep>
      <pdc>E<m>xit
        <action run=Exit>
    </abc>
```

```
    <abc><m>View

    <abc><m>Options

    <abc><m>Help
     <pdc><m>Help for help...
       <action run=helphelp>
     <pdc><m>Extended help...
       <action run=exhelp>
     <pdc><m>Keys help...
       <action run=keys>
     <pdc>Help <m>index...
       <action run=index>
     <pdc><m>Tutorial...
       <action run=tutorial>
     <pdsep>
     <pdc><m>About...
       <action run=logo>
    </abc>
  </ab>

  <info>
  </info>
</panel>
```

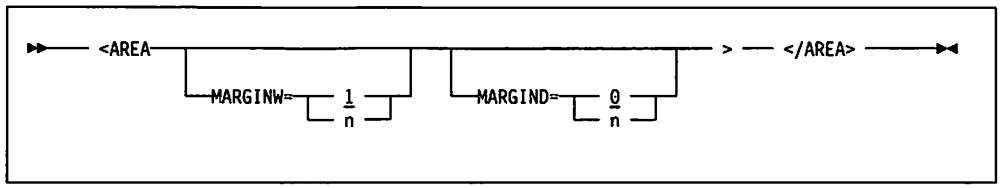
Selection Field Choice Actions: Pushbuttons also indicate immediate actions to the user. This example shows how the ACTION tag is used to define the action that occurs when a pushbutton is selected.

```
<!doctype dm system>

<panel name=action2 width=40>
  <selfld type=action dir=horiz>
     <choice>Ok
        <action run=ENTER>
     <choice>Cancel
        <action run=CANCEL>
     <choice>Help
        <action run=EXHELP>
  </selfld>
</panel>
```

AREA (Area)

The AREA tag defines the scrollable portion of a panel.



MARGINW=1 | n
The MARGINW attribute specifies a margin along the left and right sides of the panel area. This attribute allows you to specify the width of the margin in character units. The range is 1 to 32. The default is 1.

Note: If you code the MARGINW attribute on a help panel it will be ignored, and a warning message will be issued at compile time.

MARGIND=0 | n
The MARGIND attribute specifies a margin along the top and bottom of the panel area. This attribute allows you to specify the depth of the margin in character units. The range is 0 to 32. The default is 0.

Note: If you code the MARGIND attribute on a help panel, it will be ignored, and a warning message will be issued at compile time.

Description

The AREA tag defines the scrollable portion of a panel. If the entire contents of an area in an application panel are not visible, the vertical, horizontal, or both scroll bars become visible, as needed, to allow the user to view the entire contents of the panel by scrolling. Help panels always have a visible vertical scroll bar, and do not have a horizontal scroll bar.

If you do not code an AREA tag on a panel definition, the content of the panel that exceeds the panel dimensions is truncated.

Conditions

You can define only one area for each panel definition.

The following conditions apply only to help panels:

- The AREA tag is required on help panels.
- The AREA tag must be coded immediately after the HELP tag or after the last indexing tag (ICMD, ITOP or ISYN), if one is coded.
- You cannot start any tags after the AREA end tag on a help panel.

Nested Tags

You can code the following tags within the AREA tag:

Tag	Dialog Element	Usage	Page
DIVIDER (area divider)	Application Panel	Multiple	15-48
DTACOL (data column)	Application Panel	Multiple	15-56
DTAFLD (data field)	Application Panel	Multiple	15-60
INFO (information)	Application Panel Help Panel	Multiple	15-82

Tag	Dialog Element	Usage	Page
LSTFLD (list field)	Application Panel	Multiple	15-106
REGION (region)	Application Panel	Multiple	15-149
SELFLD (selection field)	Application Panel	Multiple	15-156
SELLST (selection list)	Application Panel	Multiple	15-161
UC (user control)	Application Panel	Multiple	15-173

Example

Application Panel Area: The following markup contains a scrollable area definition, which produces scroll bars if the panel is sized such that all of the panel body is not visible. The scroll bars can be used to scroll through the panel body and view its contents. Figure 15-2 and Figure 15-3 on page 15-13 show the formatted results before and after scrolling.

```
<panel name=area1 depth=12 width=50>Fine Wines
  <area>
    <divider type=none gutter=1>
    <selfld type=multi>Choose one or more types:
      <choice>Burgundy
      <choice>Sparkling Burgundy
      <choice>Rose
      <choice>Blush
      <choice>Dry
    </selfld>
    <divider type=none gutter=2>
    <region dir=horiz>
      <selfld>Choose one:
        <choice>Domestic
        <choice>Imported
      </selfld>
      <divider type=none gutter=4>
      <selfld>Choose one:
        <choice>Grape
        <choice>Apple
      </selfld>
    </region>

  </area>
</panel>
```

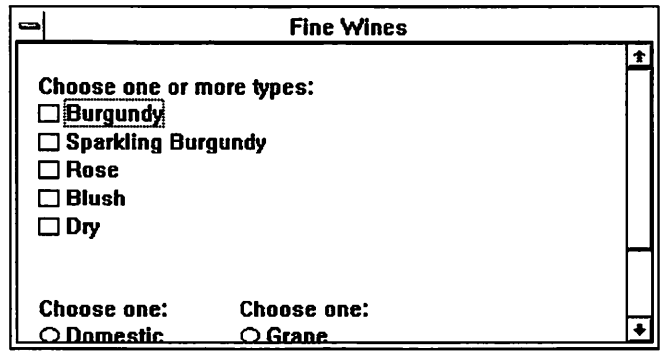


Figure 15-2. Application Panel Scrollable Area Before Scrolling

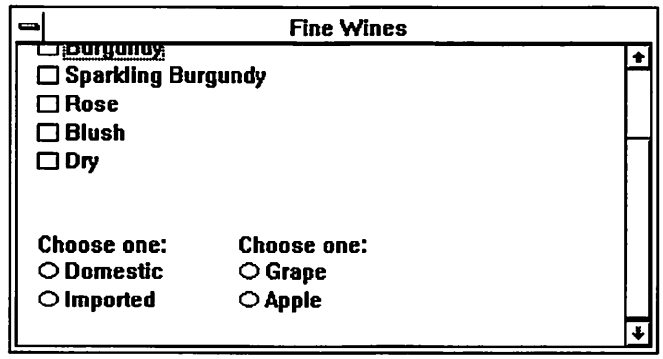
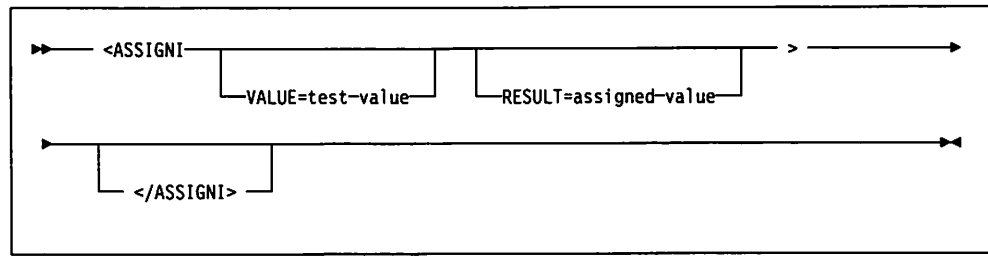


Figure 15-3. Application Panel Scrollable Area After Scrolling

ASSIGNI (Assignment List Item)

The ASSIGNI tag defines a list element in an assignment list.



VALUE = test-value
This attribute defines the value to be matched by a search argument when performing the assignment.

After the panel is submitted and all input translations are performed, the value of the data field's data variable (DATAVAR) is compared to the value of each VALUE attribute in succession until a match is found.

When the Dialog Manager finds a match, it assigns the value in the RESULT attribute to the variable named on the ASSIGNL tag. If the Dialog Manager does not find a match, no assignment is made.

If you omit this keyword, any value satisfies the test and the Dialog Manager assigns the *assigned-value* to the dialog variable.

If a *test-value* appears more than once in the list, the first occurrence is used.

RESULT = assigned-value
This attribute specifies the resulting value of the assignment if a match occurs on the *test-value* specified by VALUE.

Assigned-value specifies the character string value for the Dialog Manager to assign to the variable named on the ASSIGNL tag. If you omit this keyword, the *test-value* is assigned to the variable.

Note: The assigned value is placed directly into the pool variable and no translates or validations are performed on it.

Description

The ASSIGNI tag defines a list element in an assignment list.

Each ASSIGNI tag provides information necessary to assign a value (the RESULT attribute) to a variable (defined with the ASSIGNL tag) based on the *test-value* (the VALUE attribute) of the variable named on the DTAFLD tag. As many ASSIGNI tags as are necessary can be included within the assignment list.

If both the VALUE and the RESULT attributes are omitted, the DESTVAR attribute is assigned the value of the data field's data variable (DATAVAR).

Note: Both the *test-value* and the *assigned-value* are internal values and are not translatable.

Conditions

You must code an ASSIGNI tag within an ASSIGNL tag.

Nested Tags

None.

Example

The assignment list in the following application panel markup assigns the value specified in the associated data field to another variable, *decimal*, based on the result of a match with the nested assignment list item.

```
<!doctype dm system>

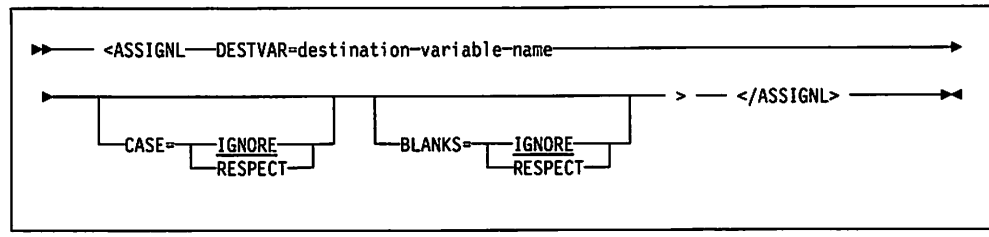
<varclass name=varcls1 type='char 1'>
</varclass>
<varclass name=varcls2 type='char 2'>
</varclass>

<varlist>
  <vardcl name=hexvar varclass=varcls1>
  <vardcl name=decimal varclass=varcls2>
</varlist>

<panel name=assgn1>Hex to Decimal
  <topinst>Enter a hexadecimal digit.
  <dtafld  datavar=hexvar  > Hexadecimal Value
    <assignl destvar=decimal>
      <assigni value=0>
      <assigni value=1>
      <assigni value=2>
      <assigni value=3>
      <assigni value=4>
      <assigni value=5>
      <assigni value=6>
      <assigni value=7>
      <assigni value=8>
      <assigni value=9>
      <assigni value=a result=10>
      <assigni value=b result=11>
      <assigni value=c result=12>
      <assigni value=d result=13>
      <assigni value=e result=14>
      <assigni value=f result=15>
      <assigni result ="?">
    </assignl>
</panel>
```

ASSIGNL (Assignment List)

The ASSIGNL tag defines an assignment list.



DESTVAR = destination-variable-name
DESTVAR is the attribute that receives the assignment value. The assignment is performed without checks and translates even if they are defined in the VARCLASS for the destination variable name. You can code multiple assignment lists if you need to assign values to additional variables.

Note: Variable names used in a DESTVAR attribute should not be used in a DATAVAR attribute for an input or input/output field elsewhere on the panel. If the same variable names are used, the assignment value would be overlaid by the value from the input or input/output field.

CASE = IGNORE | RESPECT
IGNORE means that case differences are to be ignored during comparisons when trying to match an input value of type character with a list element. Country and code page information is used to make this determination. RESPECT means that case is to be taken into account.

BLANKS = IGNORE | RESPECT
IGNORE means that the Dialog Manager strips leading blanks in the value of the panel variable before conducting the search. RESPECT means that leading blanks are significant when the Dialog Manager is considering whether the compared values match.

Description

The ASSIGNL tag defines an assignment list. ASSIGNI tags, which define the elements of the assignment list, appear within the ASSIGNL tag.

Assignment lists are optional and provide a means of assigning a value to one variable based on the content of another. The Dialog Manager compares the value of the variable specified with the DATAVAR attribute of the DTAFLD tag against the values in the ASSIGNI tags.

If more than one ASSIGNL tag is coded using the same or different DESTVAR attributes and the same test value appears in more than one list, then the first test value is the one used to determine the assigned value.

Note: ASSIGNL processing only occurs after the panel is submitted and after translates and checks on DTAFLD's data variable are performed.

Conditions

You can code an ASSIGNL tag only within a DTAFLD tag.

Nested Tags

You can code the following tag within the ASSIGNL tag:

Tag	Dialog Element	Usage	Page
ASSIGNI (assignment list item)	Application Panel	Multiple	15-14

Example

The assignment list in the following application panel markup assigns the value specified in the associated data field to another variable, *decimal*, based on the result of a match with the nested assignment list item.

```
<!doctype dm system>

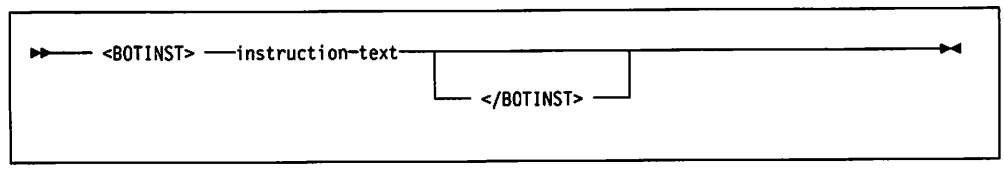
<varclass name=varcls1 type='char 1'>
</varclass>
<varclass name=varcls2 type='char 2'>
</varclass>

<varlist>
  <vardcl name=hexvar varclass=varcls1>
  <vardcl name=decimal varclass=varcls2>
</varlist>

<panel name=assgn1>Hex to Decimal
  <topinst>Enter a hexadecimal digit.
  <dtafld  datavar=hexvar  > Hexadecimal Value
    <assignl destvar=decimal>
      <assigni value=0>
      <assigni value=1>
      <assigni value=2>
      <assigni value=3>
      <assigni value=4>
      <assigni value=5>
      <assigni value=6>
      <assigni value=7>
      <assigni value=8>
      <assigni value=9>
      <assigni value=a result=10>
      <assigni value=b result=11>
      <assigni value=c result=12>
      <assigni value=d result=13>
      <assigni value=e result=14>
      <assigni value=f result=15>
      <assigni result ="?">
    </assignl>
</panel>
```

BOTINST (Bottom Instruction)

The BOTINST tag defines a bottom instruction for a panel.



Instruction-text

The *instruction-text* specifies the text to appear as a bottom instruction for the panel.

Description

The BOTINST tag defines a bottom instruction for a panel. If you want multiple instructions, use a new bottom instruction tag for each new instruction. The bottom instruction is formatted (word-wrapped) based on the width specified on the PANEL tag or a width of 128, whichever is smaller, and does not reformat when the window is sized.

Conditions

The BOTINST tag must immediately precede the CMDAREA tag. If no CMDAREA tag is specified, it must precede the PANEL end tag (</PANEL>).

The BOTINST tag cannot be nested within an AREA tag.

Bottom instructions are not scrollable.

Nested Tags

None.

Example

The following application panel markup contains two bottom instruction definitions following the scrollable area. Figure 15-4 on page 15-19 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>
</varclass>

<varlist>
  <vardcl name=red    varclass=sampcls>
  <vardcl name=blue   varclass=sampcls>
  <vardcl name=green  varclass=sampcls>
  <vardcl name=black  varclass=sampcls>
</varlist>

<panel name=botins1 help=colorh depth=10 width=30>Color Choices
  <area>
    <selfld>
    <choice checkvar=red>Red
    <choice checkvar=blue>Blue
    <choice checkvar=green>Green
    <choice checkvar=black>Black
    </selfld>
  </area>
  <botinst>Select a color, and press Enter to continue.
  <botinst>Press Esc to exit the panel.
</panel>
```

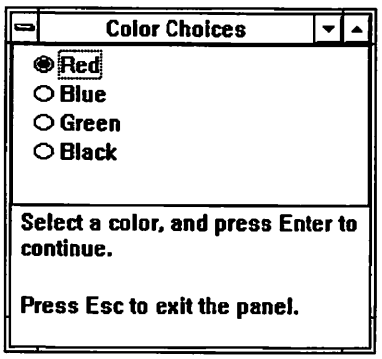
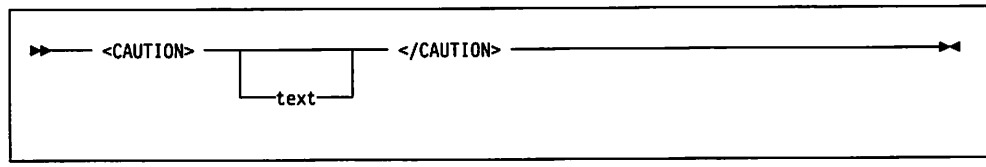


Figure 15-4. Bottom Instruction

CAUTION (Caution)

The CAUTION tag defines a statement that alerts the user of a risk.



text
This is the text of the CAUTION tag.

Description

The CAUTION tag defines a statement that alerts the user of a risk. Use a CAUTION tag to alert the user to a condition that may have serious consequences.

Code a caution statement before the text to which it pertains so that the user reads about the possible risks before reading the text. When a caution is displayed, the word "CAUTION:" or its translated equivalent appears on the screen and the caution text displays on the next line.

Conditions

You can use the CAUTION tag only within an information region. A CAUTION tag must immediately follow an LI, LP, or P tag.

Nested Tags

You can code the following tags within the CAUTION tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
XMP (example)	Application Panel Help Panel	Multiple	15-202

Examples

The following markup illustrates a CAUTION tag in an application panel. Figure 15-5 shows the formatted result.

```
<panel name=cautio1 width=50>Information on Cockatoos
  <area>
  <info>
    <h1>General Background
    <p>The Cockatoo is a native of Australia.
    The fine birds we get are carefully
    selected and warranted to
    be in superb physical health.
    <p><caution>
    These birds are native wild birds and require SOME taming.
    They are not recommended for inexperienced bird owners.
    </caution>
  </info>
  </area>
</panel>
```

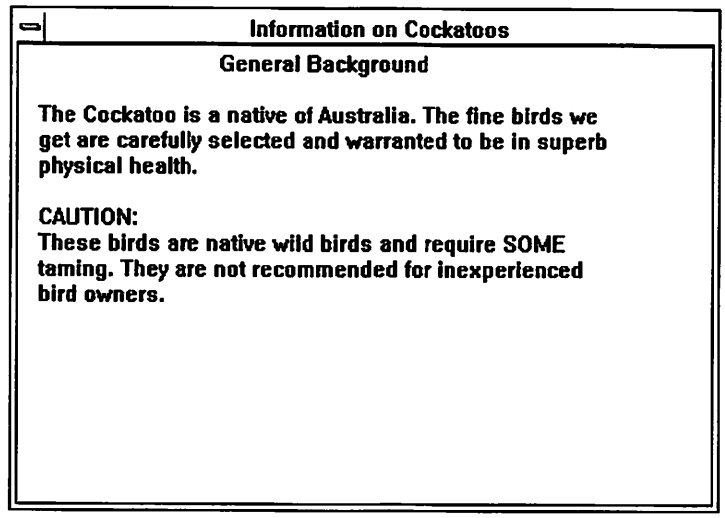
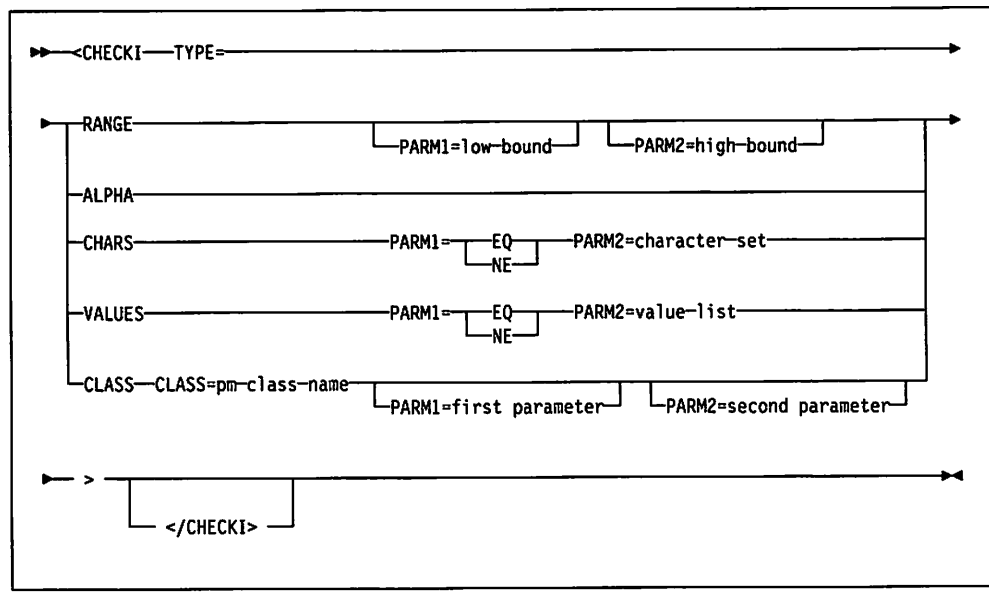


Figure 15-5. Caution Statement

CHECKI (Validity Check Item)

The CHECKI tag defines a single test of an input value.



TYPE =

The TYPE attribute specifies the type of check to be performed. The valid types are:

RANGE

This attribute allows you to check for a value within a numeric range. The specified range includes the end points. The range delimiters can include 31 digits and may also contain a decimal point (.) and sign (- or +). If no sign is coded, the value is assumed to be positive.

PARM1 = low-bound

The PARM1 attribute supplies the low bound, if any. If you do not code a value, the default is "-" (minus) followed by 31 9's (that is, -9999...99). Negative values must be coded with the minus sign on the left.

PARM2 = high-bound

The PARM2 attribute supplies the high bound, if any. If you do not code a value, the default is 31 9's (that is, 9999...99).

The value must be numeric for the check to work. Negative values must be coded with the minus sign on the left.

ALPHA

This attribute limits the character set to A – Z, a – z or the space character.

CHARS

This attribute specifies the CHARS check of characters allowed or not allowed within a string.

PARM1 = EQ | NE

The PARM1 attribute contains EQ or NE. EQ is used to determine whether the value must be one of the values specified, or must not be any of the the values specified in PARM2. NE is used to ensure that the value string does not contain any of the characters in PARM2.

PARM2 = character-set
The PARM2 attribute specifies a set of characters. The characters allowed for PARM2 are restricted to the following: A−Z, a−z, 0−9, and . , : ; ? () ' " / - _ & + % * = < > and the space character.

If a blank is to be included in the CHARS check, it must be explicitly included in the PARM2 string.

VALUES
Specifies whether the value entered must be the same as or different from one of the values specified in PARM2.

PARM1 = EQ | NE
The PARM1 attribute contains EQ or NE. EQ is used to determine whether the value must be one of the values specified in the *value-list*. NE is used to ensure that the value string must not contain any of the values specified in the *value-list*.

PARM2 = value-list
The PARM2 attribute specifies the list of values. If the list contains more than one value, it must be enclosed in quotes. If a value in the list contains blanks, it must be enclosed in single quotes if the entire list is in double quotes, or double quotes if the entire list is in single quotes. Each value in the list is separated by blanks or enclosed in quotes.

For example:

```
dog
'dog cat bird lion'
'"Advanced Topic" Index "Table of Content" Glossary'
"parsley onion 'black pepper' garlic"
```

Case is respected in a values check. If case-insensitivity is wanted, you must code a translate list of format UPPER and the values in the list in uppercase. The characters allowed for PARM2 are restricted to the following: A−Z, a−z, 0−9, and . , : ; ? () ' " / - _ & + % * = < > and the space character.

CLASS
This specifies that a check user exit will perform the check. For more information on check user exits, refer to the *Dialog Manager Guide and Reference*.

CLASS = pm-class-name
The CLASS attribute specifies the class name of the check user exit.

PARM1 = first parameter
The PARM1 attribute specifies the parameters for the exit. This string is unaltered by the compiler and the interpretation of this is determined by the check user exit.

PARM2 = second parameter
The PARM2 attribute specifies the parameters for the exit. This string is unaltered by the compiler and the interpretation of this is determined by the check user exit.

Description

The CHECKI tag defines a single test of an input value. Validity checking occurs only on input. Validity checking is bypassed if an explicit match is found on a translate list item. See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

Blanks are respected on all check items. If blanks should be ignored, place a STRIP translation before the check lists.

There can be multiple CHECKL tags within a VARCLASS tag and multiple CHECKI tags within a CHECKL tag. When you code multiple CHECKI tags within a CHECKL tag, only one must be true for the entire CHECKL to be true. When you code multiple CHECKL tags, all of them must be true for the variable value to be accepted.

Conditions

You can code a CHECKI tag only within a CHECKL tag.

The CHECKI types RANGE, ALPHA, CHARS and VALUES are incompatible with the DBCS, MIXED and ANY variables. In these cases, the CHECKI tag will be ignored.

Nested Tags

None.

Example

The check lists in the following markup show the different types of validity check items (CHECKI) and how the CHECKI tag is used on input values.

```
<!doctype dm system>

<varclass name=vname type='char 40'>
  <xlati format=upper> </xlati>
  <checkl>
    <checki type=chars Parm1=EQ Parm2='ABCDEFGHIJKLMNOPQRSTUVWXYZ .,'>
  </checkl>

<varclass name=vlevel type='NUMERIC 2'>
 <checkl>
   <checki type=range Parm1=1 Parm2=12>
 </checkl>

<varclass name=vsubject type='Char 20'>
  <xlatl format=upper></xlatl>
  <checkl>
     <checki type=values Parm1=EQ Parm2='MATH SCIENCE ENGLISH'>
  </checkl>

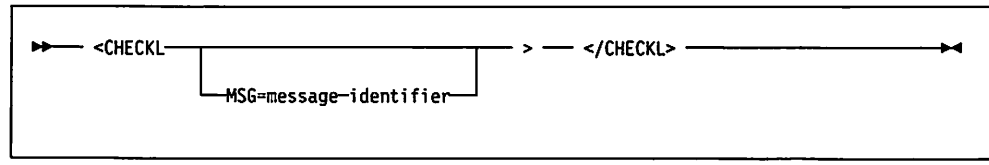
<varclass name=vgrade  Type='CHAR 1'>
  <xlatl format=upper></xlatl>
  <checkl>
    <checki Type=chars Parm1=EQ Parm2='ABCDFI'>
  </checkl>

<varlist>
  <vardcl name=name varclass=vname>
  <vardcl name=level varclass=vlevel>
  <vardcl name=subject varclass=vsubject>
  <vardcl name=grade varclass=vgrade>
</varlist>

<panel name=chckl01>Grade Report Summary
  <topinst>Complete the information below and press Enter.
  <area>
    <dtacol pmtwidth=15 Entwidth=25 Deswidth=15>
      <dtafld datavar=name>Student's name
      <dtafld datavar=level>Grade level
      <dtafld datavar=subject>Subject
      <dtafld datavar=grade>Final grade
        <dtafldd> Enter A-F or I for Incomplete.
    </dtacol>
  </area>
</panel>
```

CHECKL (Validity Check List)

The CHECKL tag defines a list of validity checks the Dialog Manager applies to the input variables.



MSG = message-identifier
The MSG attribute identifies the message to be displayed if the value fails all embedded tests. If you do not specify a message, the Dialog Manager provides a default message.

Description

The CHECKL tag defines a list of validity checks the Dialog Manager applies to the input variables. The CHECKI tags within the check list perform validation tests. If the entered value fails all of the tests, the Dialog Manager either issues the action message identified on the check list tag or (if Cancel or Exit was requested) returns an error to the application. (Refer to the "Field Validation" section of the *Dialog Manager Guide and Reference* for more information.) The action message displayed is identified on the MSG attribute of the check list tag. If the MSG attribute is not used, the Dialog Manager supplies a default message. When the value passes one of the items within a check list, no other check items within that check list are performed, but any other check lists are performed. Thus, you should code the check items in a check list in order from most likely to succeed to the least likely.

The validity checking defined by these tags occurs only on input values entered by the user. Validity checking is bypassed if an explicit match is found on a preceding inbound translation item. See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

There can be multiple CHECKL tags within a VARCLASS tag and multiple CHECKI tags within a CHECKL tag. When you code multiple CHECKI tags within a CHECKL tag, only one must be true for the entire check list to be true. When you code multiple CHECKL tags all of them must be true for the variable value to be accepted.

Conditions

CHECKL tags must be coded within the VARCLASS tag.

CHECKL tags must be coded after all XLATL tags.

Nested Tags

You can code the following tag within the CHECKL tag:

Tag	Dialog Element	Usage	Page
CHECKI (validity check)	None	Single	15-22

Example

The following markup contains check lists that perform validation for the time and date fields in the application panel. The example also shows how multiple check lists are used.

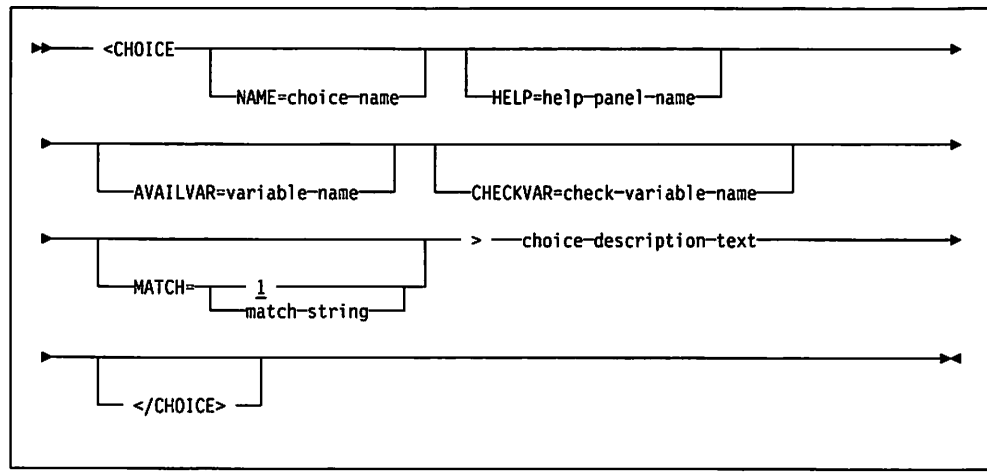
```
<!doctype dm system>

<varclass Name=vtime Type=ITIME>
  <checkl>
    <checki Type=range Parm1=0800 Parm2=1700>
  </checkl>
<varclass Name=vdate Type=IDATE>
  <checkl>
    <checki Type=range Parm1=890501 Parm2=890531>
  </checkl>
  <checkl>
    <checki Type=values Parm1=NE Parm2='890506 890507 890513 890514
       890520 890521 890527 890528 890529'>
  </checkl>
<varlist>
  <vardcl Name=time Varclass=vtime>
  <vardcl Name=date Varclass=vdate>
</varlist>

<panel Name=chckl01>May Lab Drop Off Schedule
  <topinst>Complete the information below and press Enter.
  <area>
    <dtacol Pmtwidth=15 Entwidth=15 Deswidth=15>
      <dtafld Datavar=time>Enter specimen drop-off time:
      <dtafld Datavar=date>Enter specimen drop-off date:
    </dtacol>
    <info>
      <p>You will receive a confirmation letter within two business days.
      Lab results are available one week after the drop-off date.
    </info>
  </area>
</panel>
```

CHOICE (Selection Choice)

The CHOICE tag defines information about a selection item in a selection field.



NAME = choice-name

Specifies the name of the choice.

This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the choice.
- On the ADDPOP call to position a pop-up window relative to the choice.
- On the DISPLAY call to place the cursor in the choice.
- On the PANEL definition to place the cursor in the choice.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

This attribute specifies the name of the help panel the Dialog Manager displays when the user selects help for this selection field choice.

If the user selects help on a choice, the help defined for the choice is displayed. If no help is defined for the choice, the choice will display the help panel specified on the SELFLD tag. If no help was defined for the SELFLD or CHOICE tags, the extended help panel is displayed. If an extended help panel was not defined, a message is displayed stating that no help was defined. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

AVAILVAR = variable-name

The AVAILVAR attribute specifies the availability of the choice. If the value of the specified variable name is true (1) during display, the choice is available. If the value is false (0), the choice is unavailable. The attribute value must be coded without using % notation.

If you do not specify this attribute, the choice is always available. You must ensure that the value of the named variable correctly relates to the availability of this choice. An unavailable choice is indicated by de-emphasizing the choice.

Typically, a choice for a selection field of TYPE=ACTION is implemented by issuing a command when the choice is selected. The command to issue is named by coding an ACTION tag under the CHOICE tag. Like choices, commands can also be made unavailable (through the AVAILVAR attribute on the CMD tag). If you specify an AVAILVAR attribute for both a command and a choice that references the command, be sure to keep the two synchronized.

CHECKVAR=check-variable-name
This attribute defines a variable that, for single-choice and multiple-choice selection fields, is used for the following:

- To preselect choices when the panel is initially displayed. If the value of the check variable is the same as the match string you specify with the MATCH attribute, the choice is preselected.
- To indicate to the application which choice or choices are selected by the user. This function is described in the SELFLD (selection field) tag description.

The CHECKVAR attribute value must be specified without using % notation.

This attribute is also used on immediate-action selection fields to indicate a default action pushbutton.

MATCH=1 | match-string
This attribute defines the value for the check variable that causes the choice to be preselected. It is also used to indicate to the DM application which choice or choices on a single-choice or multiple-choice selection field were chosen. The match string can be any character string. If the match string is not specified, the default is 1.

choice-description-text
This indicates the text of the choice item. The maximum length is limited to 40 bytes. If you use a mnemonic (M tag) within the *choice-description-text*, the mnemonic character is emphasized with an underscore at run time and provides fast path selection for keyboard users.

Description

The CHOICE tag defines information about a choice in a selection field. The behavior and appearance of the choice depends on which type of selection field it is: SINGLE, MULTI or ACTION.

Choices in a single-choice selection field appear as radio buttons. Selecting one of the choices causes the previously selected choice in the selection field to be de-selected.

Choices in a multiple-choice selection field appear as check boxes. The user can select one or more of the check boxes, or not select any.

Choices in an immediate-action selection field appear as pushbuttons. The ACTION tag must be coded within the CHOICE tag for an immediate action to occur when the choice is selected.

See "SELFLD (Selection Field)" on page 15-156 for more information on the behavior of choices within a selection field.

Conditions

You must code the CHOICE tag within a SELFLD tag.

To conform to CUA and have the expected interaction:

- Specify the CHECKVAR attribute.
- All choices within an immediate-action selection field or a single-choice selection field must name the same variable in the CHECKVAR attribute and have different values for the MATCH values.
- Choices within a multiple-choice selection field must name different variables in their CHECKVAR attributes.

Nested Tags

You can code the following tags within the CHOICE tag:

Tag	Dialog Element	Usage	Page
ACTION (action)	Application Panel	Multiple	15-7
M (mnemonic)	Application Panel	Single	15-110

Examples

The CHOICE tag can be used in many different ways; the following examples show some of the variations.

Single-Choice Selection Field: The following markup shows a single-choice selection field. The check variable, *sport*, will be updated with the sport elected by the user when control is returned to the application. Figure 15-6 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>
</varclass>

<varlist>
  <vardcl name=sport varclass=sampcls>
</varlist>

<panel name=choice1 width=30 depth=16>Recreation
  <selfld>Sports
    <choice checkvar=sport match=B>Baseball
    <choice checkvar=sport match=K>Basketball
    <choice checkvar=sport match=H>Hockey
    <choice checkvar=sport match=F>Football
    <choice checkvar=sport match=C>Cycling
    <choice checkvar=sport match=T>Tennis
    <choice checkvar=sport match=G>Golf
    <choice checkvar=sport match=R>Running
  </selfld>
</panel>
```

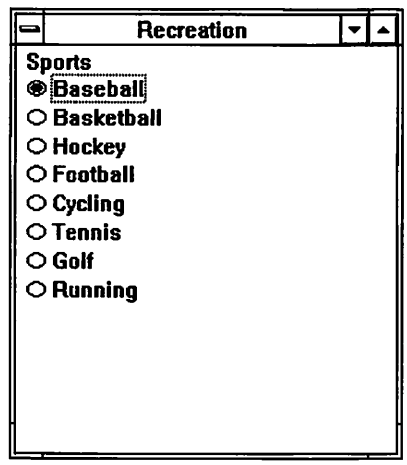


Figure 15-6. Selection Field Choice

Multiple-Choice Selection Field: The following markup shows a multiple-choice selection field. The check variables for the items chosen by the user will contain a "1" when control is returned to the application. Several items have already been chosen. Figure 15-7 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>
</varclass>

<varlist>
  <vardcl name=steer varclass=sampcls>
  <vardcl name=wheel varclass=sampcls>
  <vardcl name=brake varclass=sampcls>
  <vardcl name=radio varclass=sampcls>
  <vardcl name=ac    varclass=sampcls>
  <vardcl name=locks varclass=sampcls>
</varlist>

<panel name=choice2 depth=13 width=40>Buying an Automobile
  <selfld type=multi>Choose the options for your car:
    <choice checkvar=steer >Power steering
    <choice checkvar=wheel >4-wheel drive
    <choice checkvar=brake >Anti-lock brakes
    <choice checkvar=radio >AM/FM radio with Compact Disc player
    <choice checkvar=ac    >Air conditioning
    <choice checkvar=locks >Power windows and door locks
  </selfld>
</panel>
```

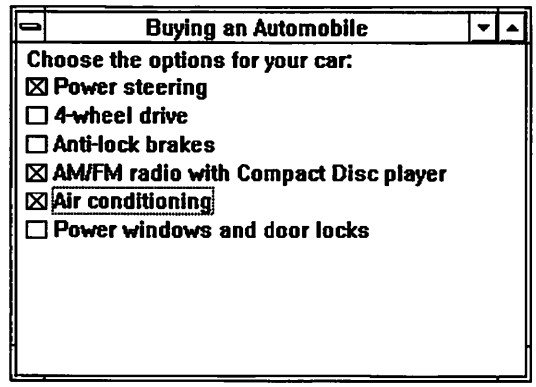


Figure 15-7. Selection Field Choice

Immediate-Action Selection Field: The following example shows an immediate-action selection field. The first choice, **Order**, is the default action. By setting the check variable, *daction*, to the match value, ORD, the **Order** pushbutton will be drawn with a heavy border when the panel is initially displayed. The heavy border indicates that ORDER is the default action, and will be selected if Enter is pressed. Figure 15-8 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type ='char 4'>
</varclass>

<varlist>
  <vardcl name=daction varclass=sampcls>
</varlist>

<panel name=choice3>
  <selfld type=action dir=horiz>
    <choice checkvar=daction match=ORD>Order
      <action run=ORDER>

    <choice>Cancel
      <action run=CANCEL>

    <choice>Help
      <action run=HELP>

  </selfld>
</panel>
```

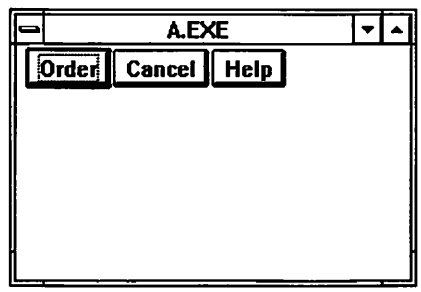
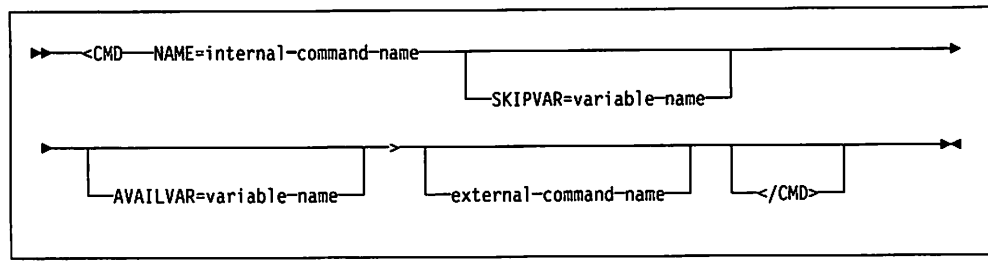


Figure 15-8. Selection Field Choice

CMD (Command Definition)

The CMD tag defines commands.



NAME = internal-command-name

This attribute specifies an internal name for this command. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

The *internal-command-name* is used in two ways:

- As the command table search criteria when a key defined in the current key mapping list is selected or when a pull-down choice or selection field choice with an associated RUN action is selected.
- As the value passed to applications when the command action is PASSTHRU or SETVERB. See "CMDACT (Command Action)" on page 15-37 for more information on the PASSTHRU and SETVERB command actions.

SKIPVAR = variable-name

The SKIPVAR attribute specifies the name of a variable that indicates whether the command definition should be ignored or not. If, at run time, the named variable has a value of true (1), the Dialog Manager ignores the command entry and continues searching the command table for the command. If the value is false (0), the Dialog Manager stops the command table search and performs the action associated with the command. The value coded must be a variable name without the % notation.

If you do not specify this attribute, the Dialog Manager will not skip the command entry.

SKIPVAR takes precedence over the AVAILVAR attribute.

You use the SKIPVAR attribute on an application command to temporarily override another command. For example, if you wanted to override the Dialog Manager EXHELP command, you could code the following in your application command table:

```
<cmd name=exhelp skipvar=skipcmd>
<cmdact action=...>
```

When the EXHELP command is issued and variable *skipcmd* has a value of 1, the command in your application command table is skipped. The Dialog Manager continues searching for the command, which it will find in the list of DM-provided commands, assuming no earlier definition exists. When *skipcmd*

has a value of 0, the command action associated with the EXHELP command definition in your application table is performed.

AVAILVAR = variable-name
The AVAILVAR attribute specifies the name of a variable that indicates the availability of the command. If, at run time, the named variable has a value of true (1), the Dialog Manager assumes that the command is available. If the value is false (0), the Dialog Manager assumes that the command is unavailable. The value coded must be a variable name without using % notation. Ensure that the value of the variable correctly relates to command availability.

If you do not specify this attribute, the Dialog Manager assumes the command is always available.

If a command is unavailable and the user enters the command in the command area, a message will be issued stating that the command is not available. If an action for a pull-down or selection field choice is to issue a command, and that command is unavailable, the Dialog Manager issues a message when the user selects the choice.

Command availability has no affect on the de-emphasizing of pull-down choices or selection field choices. Availability of choices is determined by the AVAILVAR attribute value on the PDC and CHOICE tags.

external-command-name
Specifies an external name for this command. The name must have the following characteristics:

- 2—64 single-byte character units in length (inclusive).
- No embedded blanks are permitted; the command name can be only one word.

Lowercase characters are translated to their uppercase equivalents.

The external command name is used as the command table search criteria when a command is issued from the panel command area. If you do not give the command an external command name, the command cannot be entered in a command entry field. It can, however, be issued by selecting a key, pull-down choice, or selection field choice (through the internal command name).

Description

The CMD tag defines commands. The defined command can be issued by an user by entering the *external-command-name* in the panel command area, or by selecting a key, pull-down choice, or selection field choice that references the command's *internal-command-name*. See "KEYI (Key Item)" on page 15-88 and "ACTION (Action)" on page 15-7 for additional information.

The action to be taken when a command is issued is defined with the CMDACT (command action) tag. See "CMDACT (Command Action)" on page 15-37 for information on defining command actions.

Conditions

The CMD tag must be coded within a CMDTBL tag.

Nested Tags

You can code the following tags within the CMD tag:

Tag	Dialog Element	Usage	Page
CMDACT (command action)	Command Table	Single	15-37
T (truncation)	Command Table	Single	15-169

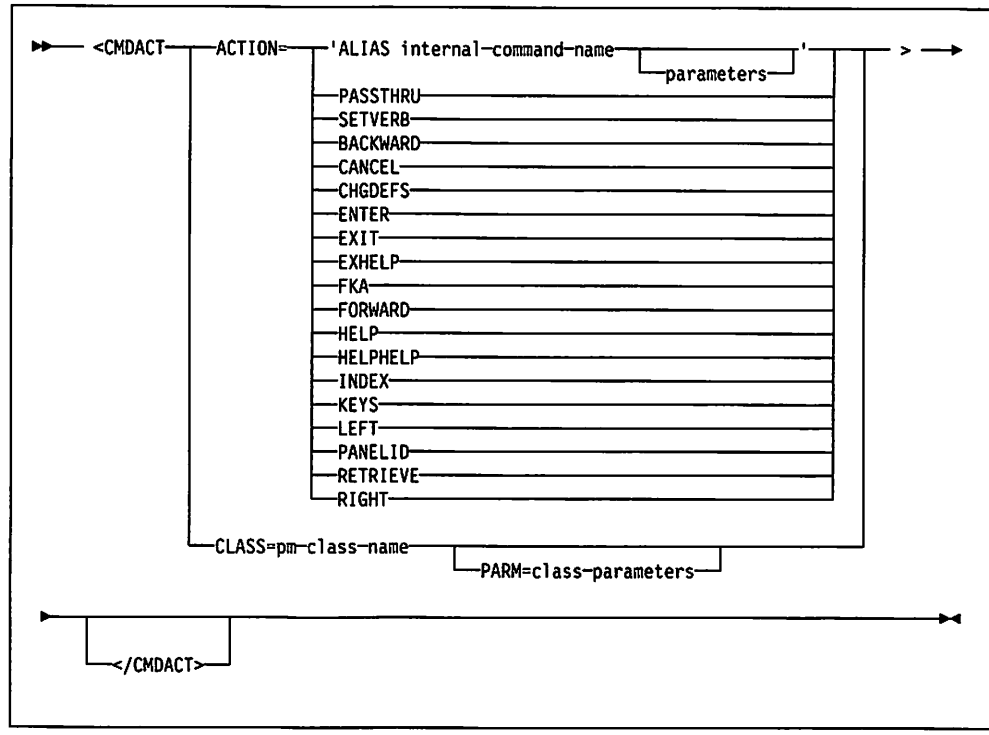
Example

The following command table markup contains two CMD definitions: Open and Close. The CMDACT tags associated with the commands specify that the commands are passed to the dialog in the ZCMD system variable.

```
<cmdtbl applid=demo>
  <cmd name=open>Open
    <cmdact action=passthru>
  <cmd name=close>Close
    <cmdact action=passthru>
</cmdtbl>
```

CMDACT (Command Action)

The CMDACT tag defines the action that will occur when an associated command is issued.



ACTION

This attribute indicates that an action processed by the Dialog Manager should be performed when the associated command is issued. The value of the attribute must be one of the following:

ALIAS internal-command-name

Provides an alternate way to express a command. For example, you might want to assign QUIT as an alias for the command EXIT.

In the command table, an alias must precede the command for which it is an alias.

You can create a chain of command aliases in a command table, as long as the result is a valid action. The last command and parameter values that the Dialog Manager encounters in the alias chain are the ones run. The command and the parameter values do not necessarily come from the same command definition entry. For example:

Command Name	Command Action
EASYKEY	ALIAS CMD PARM1 PARM2
CMD	ALIAS CMD1 PARM3
CMD1	ALIAS CMD2

In this example, if you issued the EASYKEY command, the command that would ultimately run would be CMD2 PARM3.

parameters
Any ALIAS parameters you code within the ACTION attribute value will take precedence over parameters issued from the command line. Parameters issued from a command entry field or the PARM attribute of the ACTION tag will therefore be ignored. However, if no parameters are coded within an ALIAS command, the parameters from the command entry field or the PARM attribute of the ACTION tag will be used.

PASSTHRU
The PASSTHRU attribute causes the command to be passed to the application. The *internal-command-name* and any command parameters are passed to the DM application in the ZCMD system variable.

SETVERB
This attribute provides an alternate way to pass a command to the application. The SETVERB action causes the *internal-command-name* to be passed to the DM application in the ZVERB system variable. Any command parameters are passed in the ZCMD system variable.

For further descriptions of the following attributes, refer to the *Dialog Manager Guide and Reference*.

BACKWARD
This attribute runs the BACKWARD command.

CANCEL
This attribute runs the CANCEL command.

CHGDEFS
This attribute runs the CHGDEFS command.

ENTER
This attribute runs the ENTER command.

EXIT
This attribute runs the EXIT command.

EXHELP
This attribute runs the EXHELP command.

FKA
This attribute runs the FKA command.

FORWARD
This attribute runs the FORWARD command.

HELP
This attribute runs the HELP command.

HELPHELP
This attribute runs the HELPHELP command.

INDEX
This attribute runs the INDEX command.

KEYS
This attribute runs the KEYS command.

LEFT
This attribute runs the LEFT command.

PANELID
This attribute runs the PANELID command.

RETRIEVE
This attribute runs the RETRIEVE command.

RIGHT
This attribute runs the RIGHT command.

You can specify a command action at run time by specifying the name of a variable (use % notation) for the ACTION attribute. If you specify a variable name, the Dialog Manager retrieves the action value when the command is issued. The variable value must be one of the ACTION attribute values listed above.

CLASS = pm-class-name
This attribute specifies the class name of the action user exit. It specifies that this action is supplied through an action user exit. For more information on user exits, refer to the *Dialog Manager Guide and Reference*.

PARM = class-parameters
This attribute specifies parameters for the exit. You can specify the name of a variable (using the % notation) whose value at run time will be passed as the parameter data. To specify a string that begins with a %, you must code an additional % before the string to distinguish it from a variable name. (For example, to specify the string %abc, code '%%abc' to code the variable abc, code '%abc'.)

Description

Use the CMDACT tag to define the action that will occur when the associated command is issued. You can choose from a number of processed actions (with the ACTION attribute) or by using the CLASS attribute instead of ACTION, you can define your own action.

Conditions

You can use the CMDACT tag only within the CMD tag.

Nested Tags

None.

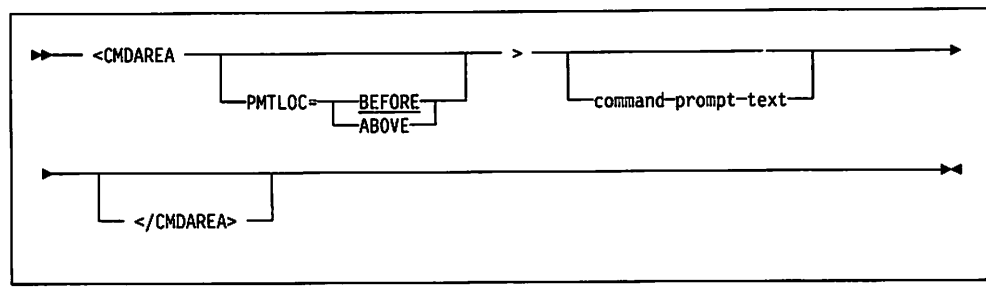
Example

The following command table markup contains three command definitions and associated command actions. The first CMDACT tag specifies a PASSTHRU action for the **Open** command. The second CMDACT tag defines an alias for the **Close** command. The third CMDACT tag contains a dialog variable that will contain the action for the **Quit** command.

```
<cmdtbl applid=demo>
  <cmd name=open>Open
    <cmdact action=passthru>
  <cmd name=close>Close
    <cmdact action='alias quit'>
  <cmd name=quit>Quit
    <cmdact action='%cmdact'>
</cmdtbl>
```

CMDAREA (Command Area)

The CMDAREA tag defines a command area, which includes the command prompt text and the command entry field.



PMTLOC = BEFORE | ABOVE

The PMTLOC attribute defines the location of the prompt text. If you specify BEFORE, the text defined by *command-prompt-text* appears on the same line of the command entry field and is located in front of that field. If you specify ABOVE, the text defined by *command-prompt-text* appears above the line of the command entry field and aligns with the command area prefix.

command-prompt-text

The *command-prompt-text* specifies the prompt text for the command area. The maximum prompt-text, without the command area prefix (===>), is 64 bytes. If the prompt text exceeds the initial panel width, it word-wraps to the next line.

The Common User Access command area prefix (===>) is added automatically in front of the command entry field.

If you do not specify the *command-prompt-text*, the default is the word Command or its translated equivalent.

Description

The CMDAREA tag defines a command area, which includes the command prompt text and the command entry field. The CMDAREA tag is primarily intended for applications that are being ported from other environments where a command line is more extensively used than in the windowing environment of OS/2.

In addition, the CMDAREA tag defines whether the prompt for the command area is on the same line as the command entry field or on the line above it. The command area is displayed at the bottom of the panel, immediately above the bottom border of the window or the function key area, if it exists.

The command entry field extends to the right side of the panel and is horizontally scrollable. It can accept up to 255 bytes of data.

Conditions

You can code the CMDAREA tag only within the PANEL tag. You can code only one command area definition for each panel.

Nested Tags

None.

Example

This following markup shows a sample panel with a command area at the bottom of the panel. Figure 15-9 shows the formatted result.

```
<!doctype dm system>

<panel name=cmdarel width=40>Command Area
  <ab>
  <abc><m>File
  <abc><m>View
  <abc><m>Options
  <abc><m>Help
  </ab>
    <topinst>Test of CMDAREA
    <area>
    <info>
    </info>
    </area>
    <cmdarea>Enter your command here
</panel>
```

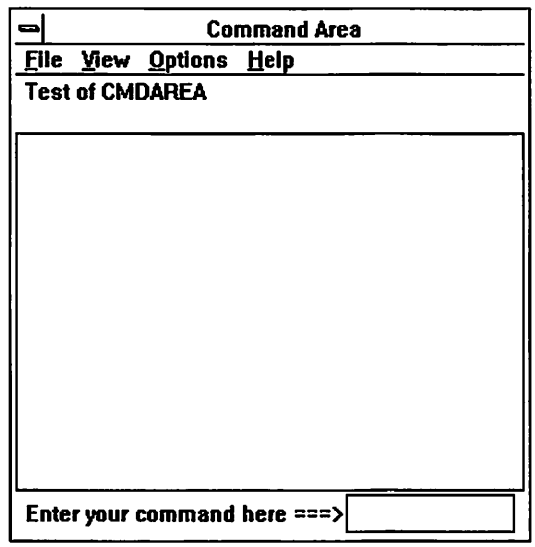
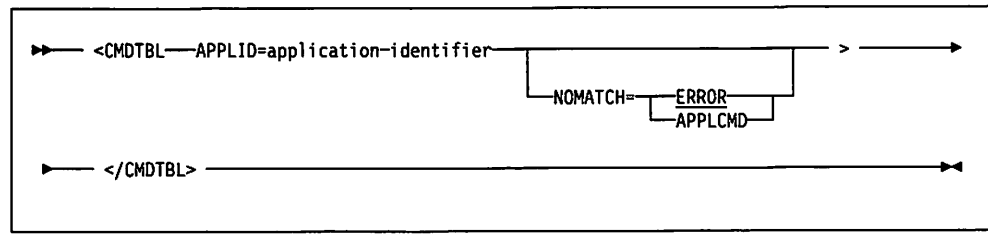


Figure 15-9. Command Area

CMDTBL (Command Table)

The CMDTBL tag provides a single place to define commands.



APPLID = application-identifier

The APPLID specifies an application identifier. This identifier is used as a prefix to the string CMDS to form the name of the command table library. Specifically, the command table library is named xxxxCMDS.DTL, where xxxx is the application identifier.

The Dialog Manager reserves the application identifier ISPx where x is any character including the space character. Do not use any of these for your application identifier.

The identifier must have the following characteristics:

- 1 – 4 characters.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

NOMATCH = ERROR | APPLCMD

This attribute specifies whether the application has any commands that are not defined to the Dialog Manager; that is, commands that are private. If you specify ERROR, any command not found within the application command table and not a DM-provided command is considered unavailable and an action message is displayed. If you specify APPLCMD, undefined commands are passed to the application for processing. In this case, it is up to the application to determine if a command is valid and, if not, to issue and display any action messages to the user.

Private commands should be used in the development stages of your application. By using private commands, you can add to and test your function without having to update your application command table. However, in the final version of your application, you should add all of your application commands to your application command table and specify NOMATCH = ERROR on the CMDTBL tag. By not allowing private commands, you relieve your application of having to distinguish between valid and undefined commands, and from having to issue an action message when an undefined command is entered by the user in a command entry field. Also, by adding all of your commands to the application command table, you can:

- Enable translation of the commands
- Prevent future conflicts between your application commands and any new DM-provided commands. If Dialog Manager added a system command that had the same name as your application command, your application command would still run if it was specified in the application command table. Otherwise, the new DM-provided command would run. The command would never be passed to your application.

Description

The CMDTBL tag provides a single place to define commands. Action and key mapping list definition tags can reference the command definitions (*internal-command-name*) within a command table.

Conditions

You cannot code the CMDTBL tag within any other tag. You can code only one command table for any application.

Nested Tags

You can code the following tag within the CMDTBL tag:

Tag	Dialog Element	Usage	Page
CMD (command definition)	Command Table	Multiple	15-34

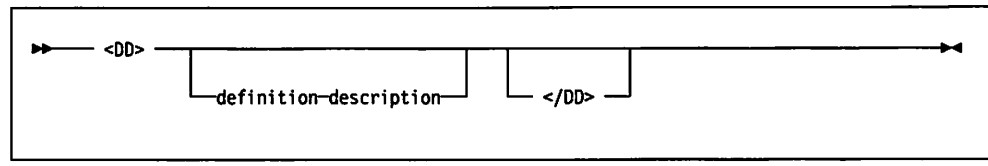
Example

The following markup shows a sample command table source file. Each command in the table has an associated command action that specifies the action taken when the command is issued.

```
<!doctype dm system>
<cmdtbl applid=demo>
  <cmd name=delete>Delete
    <cmdact action=passthru>
  <cmd name=send>Send
    <cmdact action=passthru>
  <cmd name=file>File
    <cmdact action=passthru>
  <cmd name=restore>Restore
    <cmdact action=passthru>
</cmdtbl>
```

DD (Definition Description)

The DD tag defines the description of a term in a definition list.



definition-description
This contains the text of the definition description.

Description

The DD tag defines the description of a term in a definition list.

The column in which the description starts is determined by the TSIZE attribute specified on the DL tag. See "DL (Definition List)" on page 15-50 for more information. However, if the definition term spills over the term size specified by the TSIZE attribute of the DL tag, and if the BREAK attribute of the DL tag is set to NONE, the description will start one space after its term and all other lines for that description will continue at the column specified on the TSIZE attribute. If the BREAK attribute is set to FIT, and the term spills over the term size, the description starts on the next line in the column set on the TSIZE attribute. In the case where BREAK = ALL, the description will always start on the next line in the term description column. See "DT (Definition Term)" on page 15-54 for more information.

The definition description is an implied paragraph, and can contain any text items. For example, you can insert additional paragraphs in a definition description by using the paragraph (P) tag following the description paragraph. Other tags that you want to nest within the definition list (such as OL, SL, or UL) must follow the DD tag within the list.

Conditions

Each DD tag must be paired with and follow a DT tag.

You can use the DD tag only within an information region.

The DD tags are valid only within a definition list. You can specify only one DD tag for each DT tag within a definition list (see "DL (Definition List)" on page 15-50).

Nested Tags

You can code the following tags within a DD tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel	Multiple	15-70
LINES (lines)	Application Panel	Multiple	15-95
NOTE (note)	Application Panel	Multiple	15-119
NT (note)	Application Panel	Multiple	15-121
OL (ordered list)	Application Panel	Multiple	15-123

Tag	Dialog Element	Usage	Page
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel	Multiple	15-165
UL (unordered list)	Application Panel	Multiple	15-176
XMP (example)	Application Panel	Multiple	15-202

Examples

The following help panel markup shows three definition descriptions within a definition list, each of them matched with an associated definition term. The third definition description has a paragraph nested within it. Figure 15-10 shows the formatted result.

```
<help name=dd1>Option Help
  <area>
  <info>
    <p>You can choose one of the following options:
    <dl>
      <dthd>OPTION
      <ddhd>DESCRIPTION
      <dt>Browse
      <dd>To browse a file without changing or updating.
      <dt>Update
      <dd>To place the file in update mode.
      <dt>Transfer
      <dd>To transfer the file to a specified system location.
      <p>Following transfer, the file is deleted from
         its current location.
    </dl>
  </info>
  </area>
</help>
```

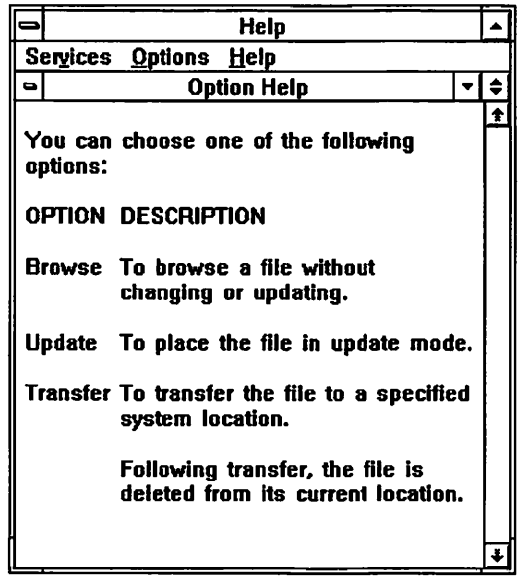
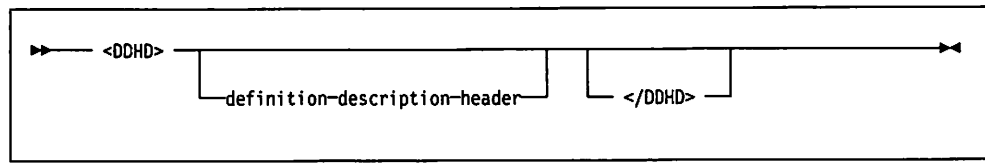


Figure 15-10. Definition Descriptions

DDHD (Definition Description Header)

The DDHD tag defines the heading for the description column of a definition list.



definition-description-header
This contains the text of the header.

Description

The DDHD tag defines the heading for the description column of a definition list. You can code multiple DDHD tags within a definition list.

Conditions

You can use the DDHD tag only within an information region.

The DDHD tag is valid only within a definition list. See "DL (Definition List)" on page 15-50 for more information on definition lists.

Each DDHD tag must be paired with a DTHD tag.

Nested Tags

You can code the following tags within a DDHD tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Examples

The DDHD tag in the following markup specifies the description header **Meaning**. Figure 15-11 shows the formatted result.

```
<help name=ddhd1>Prefix Help
  <area>
  <info>
    <p>The following list defines each of the valid prefixes.
    <dl tsize=12>
      <dthd>Prefix
      <ddhd>Meaning
      <dt>AU
      <dd>Automotive
      <dt>HB
      <dd>Health and beauty
      <dt>LG
      <dd>Lawn and garden
      <dt>SG
      <dd>Sporting goods
    </dl>
  </info>
  </area>
</help>
```

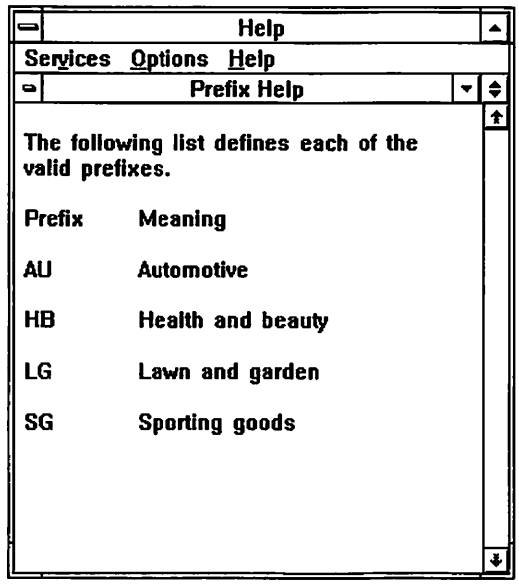
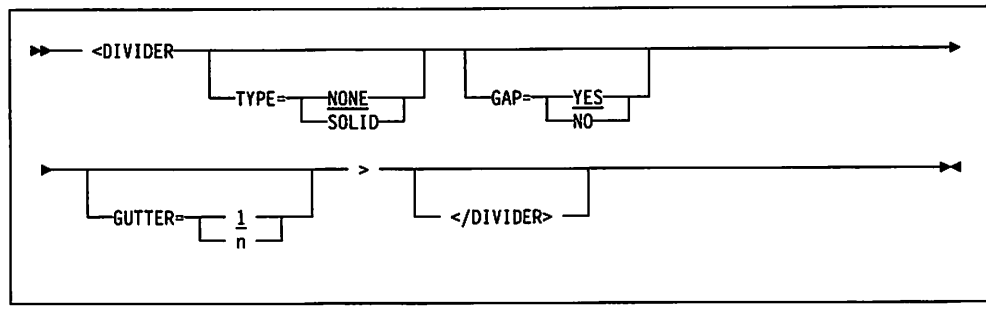


Figure 15-11. Definition Description Header

DIVIDER (Area Divider)

The DIVIDER tag defines a blank or solid divider within an area or region.



TYPE = NONE | SOLID
This attribute specifies the type of divider line.

The line width is one pixel. NONE is the default. If NONE is specified, Dialog Manager provides a blank space.

GAP = YES | NO
When GAP = NO, the divider line completely crosses from one side of the enclosing region to the other. When GAP = YES, a 1-character-unit gap remains at each end of the divider line.

GUTTER = 1 | n
This attribute specifies the total width of the divider in character units. The gutter size is a measure of the space that will be added on either side of the divider line. For example, if the GUTTER value is 4, then 2 character units of space are reserved on either side of the divider line.

The divider line can be either horizontal or vertical.

Description

The DIVIDER tag defines a blank or solid divider within an area or region. The direction of the divider line is determined by the direction of the region it is coded within. In a vertical region, the line is horizontal. In a horizontal region, the line is vertical. See "REGION (Region)" on page 15-149 for more information.

Conditions

You can code this tag within the AREA, DTACOL, PANEL and REGION tags within application panels.

Nested Tags

None.

Examples

The following markup shows three dividers. The first is a horizontal divider that is a blank space. The second divider is used to add blank space. It is also horizontal and has a gutter size of 2. The third divider is vertical because it is inside a horizontal region. It has a gutter size of 4, and is solid with a gap equal to *NO*. Notice that the dividers assume the direction opposite of the region in which they are contained. Figure 15-12 on page 15-49 shows the formatted result.

```
<!doctype dm system>

<panel name=divide1 width=40>Application Name
  <divider>
  <selfld>Colors
    <choice>Blue
    <choice>Red
    <choice>Yellow
    <choice>Green
    <choice>Purple
    <choice>Orange
  </selfld>
  <divider gutter=2>
  <region dir=horiz>
    <selfld>Shapes
      <choice>Square
      <choice>Round
      <choice>Rectangular
      <choice>Triangular
    </selfld>
    <divider type=solid gutter=4 gap=no>
    <selfld>Sizes
      <choice>Small
      <choice>Medium
      <choice>Large
      <choice>Extra large
    </selfld>
  </region>
</panel>
```

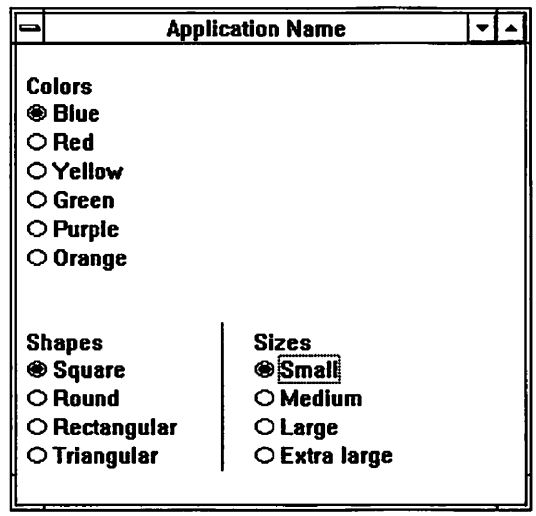
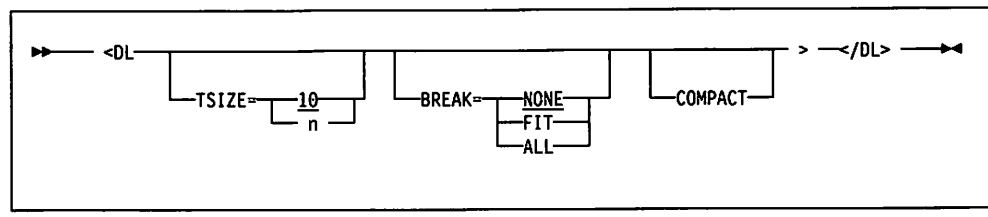


Figure 15-12. Dividers

DL (Definition List)

The DL tag defines a list of words or phrases and their corresponding definitions.



TSIZE = 10 | n

The TSIZE attribute specifies the space to be allocated for the definition list terms and term headings. The default is 10 character units. The minimum term size is 1 and the maximum is 128. If the TSIZE value exceeds the current size of the formatting area (the space between the current left and right margins), the TSIZE is assigned the current formatting area size and a warning message is issued.

BREAK = NONE | FIT | ALL

This controls the formatting of the definition terms and descriptions. If BREAK = NONE, the term is on the same line as the description, spilling into the description area if the length exceeds TSIZE; the description will start one space after the term. If BREAK = FIT, the description is on the line below the term if the term is longer than the term-size value.

In the case where BREAK = ALL, the description will always start on the next line in the term-size column.

COMPACT

The COMPACT attribute causes the list to format without a blank line between the items.

Note: The BREAK and COMPACT attributes do not affect the headings specified by the DTHD and the DDHD tags. However, the COMPACT attribute removes the line between the headings and the items in the list.

Description

The DL tag defines a list of words or phrases and their corresponding definitions. You use the DT and DD tags to identify the terms that you are defining and their descriptions, respectively. To define headings for the term and description columns in definition lists, you precede the DT and the DD tags with DTHD and DDHD tags.

If you do not explicitly set term size, the space allocated for the term size is 10 character units. If any term or term heading is longer than 10 character units, it will extend into the description or description heading line respectively. If the term is still too long to fit, it will wrap to the next line.

Conditions

You can use the DL tag only within an information region.

All DTHD and DDHD tags must precede the DT and DD tags.

Nested Tags

You can code the following tags within a DL tag:

Tag	Dialog Element	Usage	Page
DTHD (definition term header)	Application Panel Help Panel	Multiple	15-68
DDHD (definition description header)	Application Panel Help Panel	Multiple	15-46
DT (definition term)	Application Panel Help Panel	Multiple	15-54
DD (definition description)	Application Panel Help Panel	Multiple	15-44

Examples

The following help panel markup contains a compact definition list that specifies the default BREAK value of *none*, which formats the definition descriptions on the same line as the associated terms. Definition term and description headers are also included. Figure 15-13 on page 15-52 shows the formatted result. Figure 15-14 on page 15-52 shows the same definition list with a BREAK value of *fit* and the formatted result. Figure 15-15 on page 15-53 shows the same definition list with a BREAK value of *all* and the formatted result.

```
<help name=dl1>Employee Code Help
  <area>
  <info>
    <p>The following list defines the valid employee codes.
    <dl compact tsize=11 break=none>
      <dthd>Code
      <ddhd>Meaning
      <dt>Full-time
      <dd>Indicates that the employee works a
      regular schedule of forty hours or more weekly.
      <dt>Part-time
      <dd>Indicates that the employee works a regular
      schedule of twenty to forty hours weekly.
      <dt>Supplemental
      <dd>Indicates that the employee works less than
      twenty hours weekly.
      No regular schedule is in place.
    </dl>
  </info>
  </area>
</help>
```

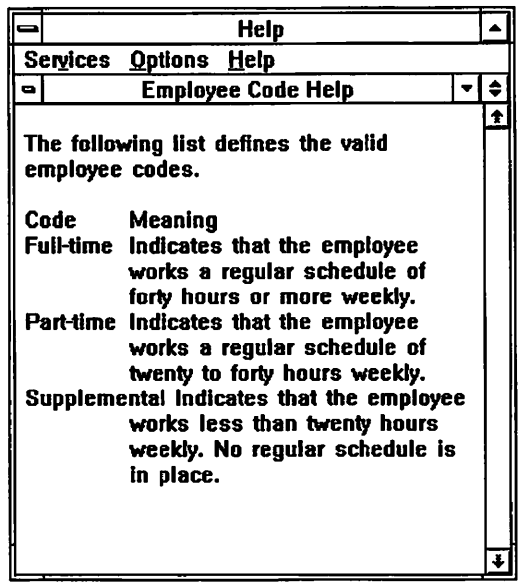


Figure 15-13. Definition List with BREAK=NONE

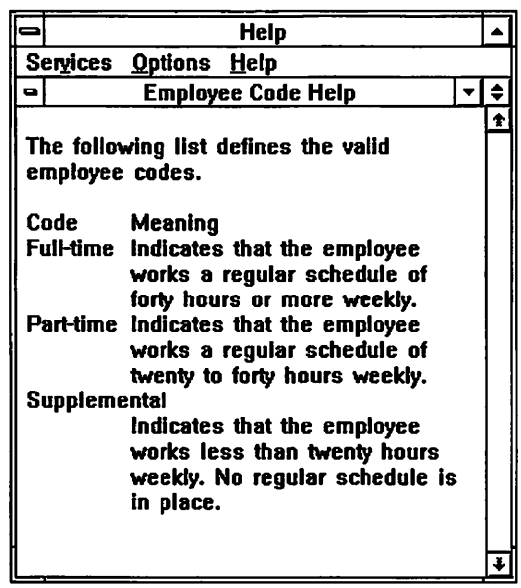


Figure 15-14. Definition List with BREAK=FIT

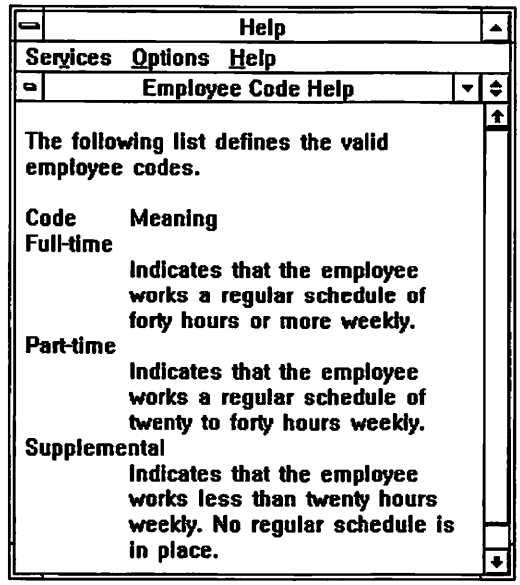
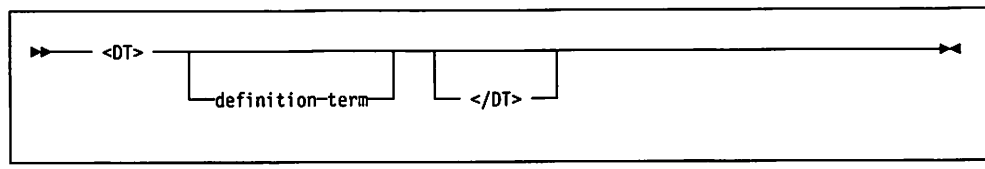


Figure 15-15. Definition List with BREAK = ALL

DT (Definition Term)

The DT tag defines the term being defined in a definition list.



definition-term
The text of the definition term.

Description

The DT tag defines the term being defined in a definition list. You can code multiple DT tags within a definition list.

Conditions

You can use the DT tag only within an information region.

The DT tags are valid only within a definition list. See "DL (Definition List)" on page 15-50 for more information.

Nested Tags

You can code the following tag within the DT tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Examples

This help panel example shows three definition terms within a definition list, each of them matched with an associated definition description. Figure 15-16 on page 15-55 shows the formatted result.

```
<help name=dtl>Option Help
  <area>
  <info>
    <p>You can choose one of the following options:
    <dl>
      <dthd>OPTION
      <ddhd>DESCRIPTION
      <dt>Browse
      <dd>To browse a file without changing or updating.
      <dt>Update
      <dd>To place the file in update mode.
      <dt>Transfer
      <dd>To transfer the file to a specified system location.
      <p>Following transfer, the file is deleted from
      its current location.
    </dl>
  </info>
  </area>
</help>
```

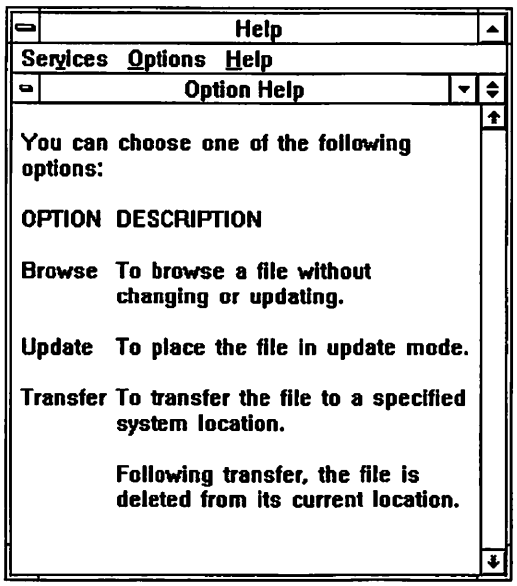
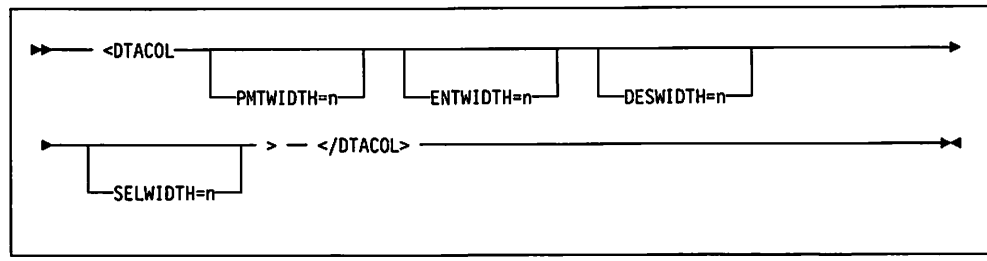


Figure 15-16. Definition Terms

DTACOL (Data Column)

The DTACOL tag defines column widths for the data fields (DTAFLD) and selection fields (SELFLD) that are coded within a DTACOL tag.



PMTWIDTH = n

The *prompt-width* attribute sets the width, in character units, of the prompts for data fields and selection fields coded within the data column. The minimum width value is 0 and the maximum is 128. If the prompt text has fewer character units than the field allows, the Dialog Manager supplies blanks to fill the remaining spaces. If any prompt contains more character units than the width you specify, that prompt will be word-wrapped to fit on multiple lines. If any field defines the PMTWIDTH attribute, it overrides the values on this attribute.

ENTWIDTH = n

The ENTWIDTH attribute specifies the width, in character units, of the data fields. The minimum width value is 1 and the maximum is 128. If any field defines the ENTWIDTH attribute, it overrides the value specified in the DTACOL tag.

Note: The ENTWIDTH attribute does not apply to nested selection fields.

DESWIDTH = n

The DESWIDTH attribute specifies the width, in character units, of the description text of the enclosed DTAFLDD tags. The minimum width value is 0 and the maximum is 128. If any field defines the DESWIDTH attribute, it overrides the value specified in the DTACOL tag.

Note: That the DESWIDTH attribute does not apply to nested selection fields.

SELWIDTH = n

The SELWIDTH attribute specifies the space to be taken up by the selection-width part of the enclosed selection fields. This is useful for getting the choices in the selection to occupy the same amount of space. The minimum width value is 1 and the maximum is 128. If you specify the SELWIDTH attribute on the SELFLD tag, it overrides the SELWIDTH specified in the DTACOL tag.

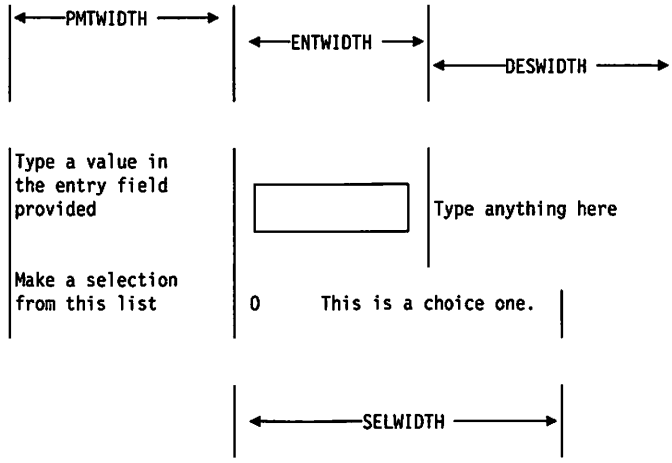
The selection part of a selection field includes either a check box or a radio button and the following text. For immediate-action selection fields, it includes the text and the surrounding pushbutton border.

Description

The DTACOL tag defines column widths for the data fields (DTAFLD) and selection fields (SELFLD) that are coded within a DTACOL tag. This allows the Dialog Manager to align the data fields and supply blanks between the prompt text, the entry field and between the entry field and data description. All fields located between the DTACOL tag and its matching end tag are aligned with these values.

If you use this tag for output fields, the fields are aligned and blanks are provided following the output fields. If you do not use this tag, no alignment of fields is done for either entry or output fields.

Fields are laid out within the data column along boundaries established by the attributes coded on this tag. The following example shows those boundaries:



The prompt-width is used by all enclosed fields. The entry-width and description-width are only used by the data fields. The selection-width is only used by selection fields. If the prompt text has fewer character units than the specified prompt-width, the Dialog Manager supplies blanks to fill the remaining spaces. When the choice text contains more character units than will fit within SELWIDTH, the excess characters are truncated.

Conditions

You can code the DTACOL tag anywhere within the panel area, but the tag must precede any DTAFLD or SELFLD tag to which it applies.

Nested Tags

You can code the following tags within the DTACOL tag:

Tag	Dialog Element	Usage	Page
DIVIDER (area divider)	Application Panel	Multiple	15-48
DTAFLD (data field)	Application Panel	Multiple	15-60
SELFLD (selection field)	Application Panel	Multiple	15-156

Example

The following markup shows a data column that is used to define the default values for prompt-width, entry-width, selection-width, and description-width. Any of the fields coded within the data column can override the values specified on the DTACOL tag by defining them explicitly on the DTAFLD or SELFLD tag.
Figure 15-17 on page 15-59 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls  type = 'char 20'>

<varlist>
  <vardcl name=name     varclass=sampcls>
  <vardcl name=co       varclass=sampcls>
  <vardcl name=ccname   varclass=sampcls>
  <vardcl name=cctype   varclass=sampcls>
  <vardcl name=ccnum    varclass=sampcls>
  <vardcl name=whereto  varclass=sampcls>
</varlist>

<panel name=dtacol1 depth=25 width=50>Airline Reservations
  <area>
    <dtacol pmtwidth=12 entwidth=12 selwidth=17 deswidth=20>

      <dtafld datavar=name required=yes>Name
      <dtafld datavar=co>Company

      <selfld pmtloc=before>Method of payment
        <choice>Cash
        <choice>Check
        <choice>Credit card
      </selfld>
```

```
      <dtafld datavar=ccname>Name on the credit card
        <dtafldd>Leave blank if the same as above.
      <dtafld datavar=cctype>Type of credit card
        <dtafldd>Visa, Mastercard, other.
      <dtafld datavar=ccnum>Credit card number
        <dtafldd>Omit dashes (-) and spaces ( )

      <divider>

      <dtafld datavar=whereto>Destination
        <dtafldd>Include country if other than the USA.

      <selfld pmtloc=before>Choose one:
        <choice>One-way ticket
        <choice>Round trip ticket
      </selfld>
    </dtacol>
  </area>
</panel>
```

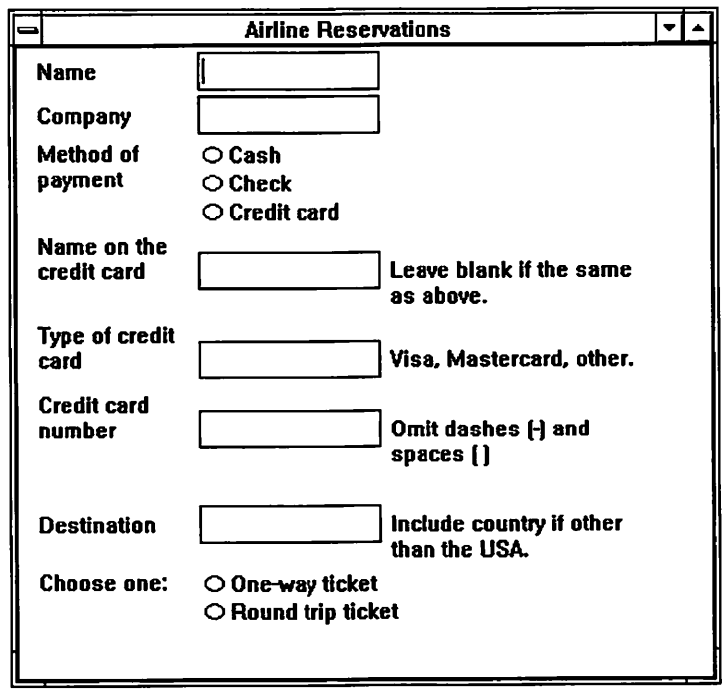
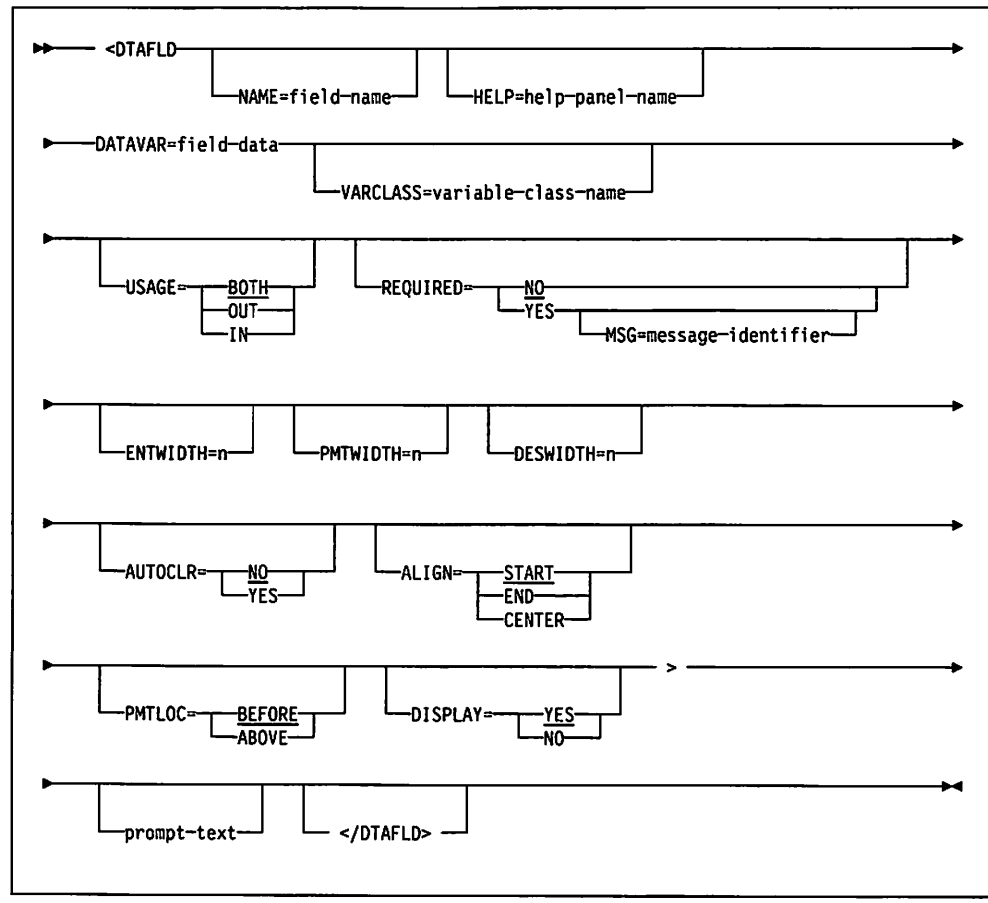


Figure 15-17. Data Column

DTAFLD (Data Field)

The DTAFLD tag defines an input field, an output field or an input/output field.



NAME = field-name

Specifies the name of the field.

This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the data field.
- On the ADDPOP call to position a pop-up window relative to the data field.
- On the DISPLAY call to place the cursor in the data field.
- On the PANEL definition to place the cursor in the data field.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

This attribute assigns a name to a help panel the Dialog Manager displays when the user selects help for this data field. If the user selects help on a data field, the help defined for the data field displays. If no help is specified for the data field, the extended help panel displays. If an extended help panel was not defined, a message is displayed stating that no help is defined. This attribute

is valid only when USAGE=IN or BOTH. The *help-panel-name* must have the following characteristics:

- 1–8 characters in length.
- The first (or only) character must be A–Z or a–z.
- Remaining characters, if any, must be A–Z, a–z, or 0–9.

Lowercase characters are translated to their uppercase equivalents.

DATAVAR=field-data
A variable name that specifies the data in the field. The value coded must be a variable-name without using % notation. The DATAVAR attribute must be a scalar variable.

The maximum length of a data value that can be displayed or entered (with possible scrolling) within a DTAFLD entry is determined by the characteristics of:

- The explicitly specified VARCLASS, (if any) or
- The default VARCLASS for the DATAVAR.

VARCLASS=variable-class-name
This attribute specifies the name of the variable class, defined using the VARCLASS tag, that overrides the default variable class referenced by the VARDCL that declared the data variable for this field. The type information of the specified variable class must be the same as the default VARCLASS for DATAVAR.

If the VARCLASS specified does not have TYPE information that matches the type information of the overridden VARCLASS, a warning message is issued.

USAGE=BOTH | IN | OUT
The USAGE attribute indicates whether the field is for input-only, output-only, or both.

An entry field is a field on a panel into which the user can enter data. An input-only field does not initially display the value of the variable associated with the field. BOTH (input/output) field, however, functions much like an input-only field but the current value of the DATAVAR is also displayed in the entry field when the panel is initially displayed. Input fields can only be one line, but may be horizontally scrollable.

An output-only field simply displays the current value of the variable associated with the field. If the text does not completely fit into the width of the output field, the output is word-wrapped to the next line until all of the text is displayed. An output-only field cannot contain the cursor.

REQUIRED=NO | YES
This attribute indicates if the field requires input. The default is NO. If you set both REQUIRED=YES and USAGE=OUT, the compiler issues a warning message and REQUIRED=YES is ignored. This attribute is valid only when USAGE=IN or BOTH.

MSG=message-identifier
The MSG attribute specifies the identifier for a message that the Dialog Manager displays when the user does not complete a required entry (REQUIRED attribute). If you do not specify a *message-identifier*, the Dialog Manager uses a default message.

This attribute is valid only if REQUIRED=YES.

ENTWIDTH = n

The ENTWIDTH attribute serves as a boundary to the data field size. The data field is left-justified within the entry column.

The range is 1 to 128.

If the ENTWIDTH attribute is not specified, the ENTWIDTH default is determined as follows:

If there is an enclosing DTACOL tag that specifies a value for the ENTWIDTH attribute, that value is used as the default. Otherwise, the default is the number of bytes defined by the associated VARCLASS, or 25 character units, whichever is smaller.

If the number of bytes defined by the associated VARCLASS exceeds the data field size, one of the following actions will occur:

- For input/both type fields, the data field is made scrollable up to the number of characters taken from the associated VARCLASS.
- For output-type fields, the variable data is wrapped to multiple lines.

PMTWIDTH = n

The prompt-width, in character units, of the prompt for the data field. If any prompt is longer than the width you specify it will be word-wrapped to fit on multiple lines. If you do not specify this attribute, the PMTWIDTH default is determined by using the value set on the PMTWIDTH attribute of any enclosing DTACOL tag as the default. Otherwise, the default is the length of the prompt text taken as a width in character units or 128, whichever is smaller. The range is 0 to 128.

If the specified PMTWIDTH is zero and text is present, a compile warning message is issued and the text is omitted.

DESWIDTH = n

The description-width, in character units, of the description text of the enclosed DTAFLDD tags. If the text is longer than this width, that text is word-wrapped to fit on multiple lines. If the specified DESWIDTH is zero and text is present, a compiler warning message is issued and the text is omitted. The range is 0 to 128.

If no DESWIDTH is specified on either the DTAFLD or DTACOL tags, the default width is the length of the description text in character units, or 128, whichever is smaller.

If specified, this attribute overrides the DESWIDTH attribute of the DTACOL tag.

AUTOCLR = NO | YES

The *auto-clear* attribute specifies whether the field will be cleared on input of new data. When the user types the first character in an auto-clear field, the Dialog Manager erases any remaining contents in the field. This attribute is valid only when USAGE = IN or BOTH. If you set both AUTOCLR = YES and USAGE = OUT, the compiler issues a warning message and AUTOCLR = YES is ignored.

ALIGN = START | END | CENTER

This attribute specifies the alignment of the variable value within the display field after all translations have been performed. Use this attribute to align the value with the start, the end, or the center of the display field.

If you use the START alignment attribute and the value being displayed is longer than the width of the field, the value is left-justified and the field is horizontally scrollable.

PMTLOC = BEFORE | ABOVE

The *prompt-location* attribute specifies whether the *prompt-text* is to occur above or in front of the data field. The following shows how the *prompt-text* will display if PMTLOC = ABOVE:



The following shows how the prompt text will display if PMTLOC = BEFORE:



DISPLAY = YES | NO

This attribute specifies whether data will display on the screen when the user types it. If you specify NO, the data will not display. This is useful when creating fields for passwords or other information which the user does not want to appear on the screen.

prompt-text

The *prompt-text* attribute specifies the prompt-text for the field. The prompt-text appears in front of or above the field, depending on the setting of the PMTLOC attribute. If you do not specify prompt text, no text appears for the field.

If the prompt-text exceeds the width defined for a prompt, it will be word-wrapped to multiple lines.

Description

The DTAFLD tag defines an input field, an output field or an input/output field.

DTAFLDD tags provide the descriptions within the data field. To insert blank lines between multiple data fields, insert a DIVIDER tag between the DTAFLD tags.

Conditions

None.

Nested Tags

You can code the following tags within the DTAFLD tag:

Tag	Dialog Element	Usage	Page
DTAFLDD (data field description)	Application Panel	Multiple	15-66
ASSIGNL (assignment list)	Application Panel	Multiple	15-16

Example

This following markup shows 13 data fields. The **Manager**, **Owner**, and **Chef** data fields require that the user enter data into that field before the Dialog Manager accepts the panel as valid. The salesperson name and number are displayed in output-only data fields. A data field description is also shown on the **Catering Director** field. Data columns are used to align the prompts and entry fields. Figure 15-18 on page 15-65 shows the formatted result.

```
<!doctype dm system>

<varclass name=numcls   type = 'char 5'>
<varclass name=sampcls  type = 'char 20'>
<varclass name=statcls  type = 'char 2'>
<varclass name=phonecls type = 'char 12'>

<varlist>
  <vardcl name=num      varclass=numcls>
  <vardcl name=name     varclass=sampcls>
  <vardcl name=addr     varclass=sampcls>
  <vardcl name=city     varclass=sampcls>
  <vardcl name=stat     varclass=statcls>
  <vardcl name=zip      varclass=numcls>
  <vardcl name=phone    varclass=phonecls>
  <vardcl name=mgr      varclass=sampcls>
  <vardcl name=own      varclass=sampcls>
  <vardcl name=chef     varclass=sampcls>
  <vardcl name=cdir     varclass=sampcls>
  <vardcl name=sales    varclass=sampcls>
  <vardcl name=salesnum varclass=numcls>
</varlist>

<panel name=dtafld1 depth=27 width=65>Wholesale Food Distributors
  <topinst>Update the customer information that has changed.
  <area>
    <dtacol pmtwidth=17 entwidth=12>
      <dtafld datavar=num>Customer Number
      <dtafld datavar=name>Name
      <dtafld datavar=addr>Address
    </dtacol>
    <region dir=horiz>
      <dtafld datavar=city>City
      <dtafld datavar=stat>State
      <dtafld datavar=zip>Zip Code
    </region>
    <dtacol pmtwidth=17 entwidth=10>
      <dtafld datavar=phone>Phone Number
      <divider>
      <dtafld datavar=mgr  required=yes>Manager
      <dtafld datavar=own  required=yes>Owner
      <dtafld datavar=chef required=yes>Chef
      <dtafld datavar=cdir >Catering Director
        <dtafldd>Leave blank if same as chef.
      <divider type=solid>
      <dtafld datavar=sales     usage=out>Salesperson Name:
      <dtafld datavar=salesnum  usage=out>Salesperson Number:
    </dtacol>
  </area>
</panel>
```

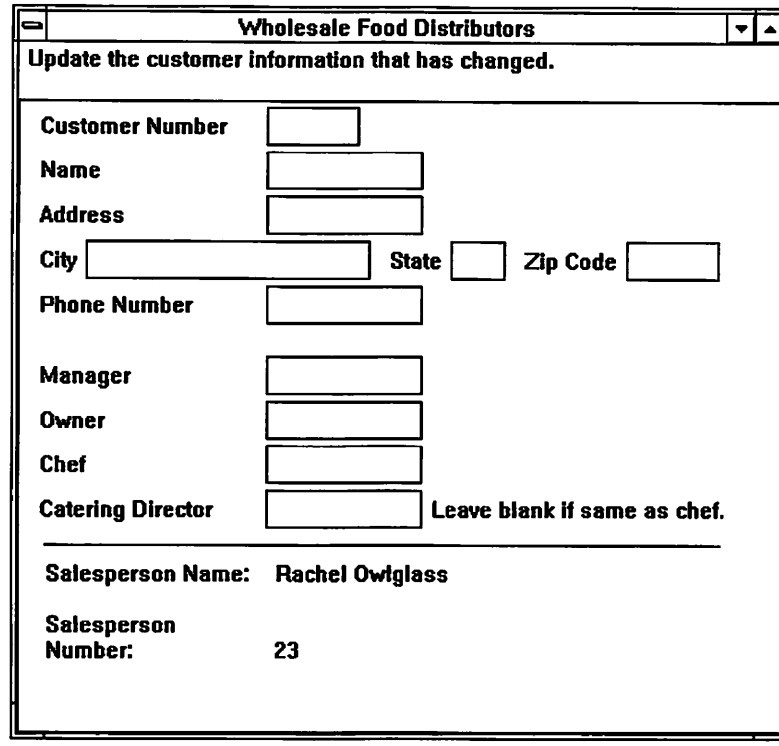
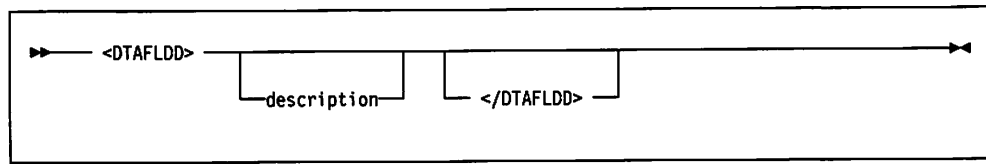


Figure 15-18. Data Fields

DTAFLDD (Data Field Description)

The DTAFLDD tag defines additional descriptive text that is associated with a data field.



description
This allows you to add descriptive text that is associated with the data field.

Description

The DTAFLDD tag defines additional descriptive text that is associated with a data field. For example, it could explain what the user can type into the field. The text will appear in the area defined by the DESWIDTH attribute of the DTAFLD tag.

You can specify more than one DTAFLDD tag for a given field. Each data field description starts a new line. For more information, see "DTAFLD (Data Field)" on page 15-60 and "DTACOL (Data Column)" on page 15-56.

Conditions

You must code the data field description tag within the DTAFLD tag that defines the field you are describing.

Nested Tags

None.

Example

This example shows a data field description on the **Filename** and **Number of copies** fields. Figure 15-19 on page 15-67 shows the formatted result.

```
<!doctype dm system>

<varclass name=vc1 type = 'char 50'>
<varclass name=vc2 type = 'char 12'>
<varclass name=vc3 type = 'char 2'>

<varlist>
  <vardcl name=curdir varclass=vc1>
  <vardcl name=fname  varclass=vc2>
  <vardcl name=copies varclass=vc3>
</varlist>
```

```
<panel name=dtfldd1 width=50 depth=10>Print
  <dtafld datavar=curdir usage=out>Directory:
  <dtafld datavar=fname>Filename
    <dtafldd>Specify the file extension.
  <dtafld datavar=copies>Number of copies
    <dtafldd>(1 - 99)
</panel>
```

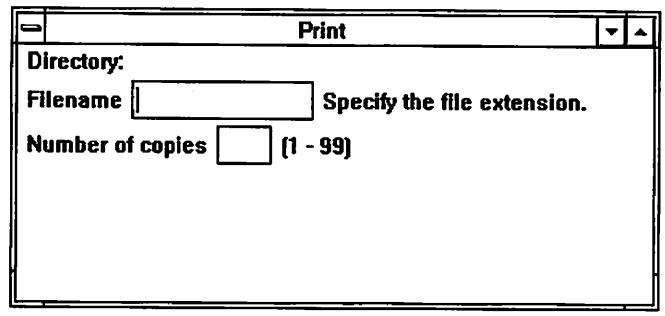
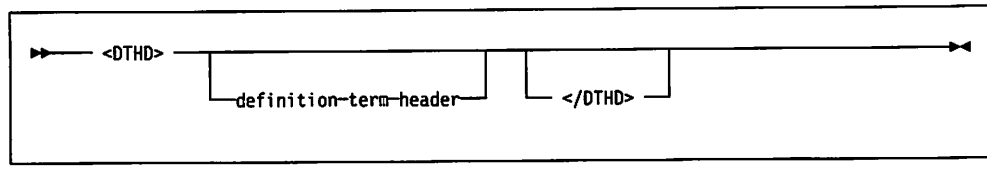


Figure 15-19. Data Field Descriptions

DTHD (Definition Term Header)

The DTHD tag defines the heading for the term column of a definition list.



definition-term-header
Indicates the text of the definition term header.

Description

The DTHD tag defines the heading for the term column of a definition list.

You can code multiple DTHD tags within a definition list.

Conditions

You can use the DTHD tag only within an information region. The DTHD tag is valid only within a definition list (see "DL (Definition List)" on page 15-50).

Nested Tags

You can code the following tag within the DTHD tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Example

The DTHD tag in the following markup specifies the definition term header prefix. Figure 15-20 on page 15-69 shows the formatted result.

```
<help name=dthd1>Prefix Help
  <area>
  <info>
    <p>The following list defines each of the valid prefixes.
    <dl>
      <dthd>Prefix
      <ddhd>Meaning
      <dt>AU
      <dd>Automotive
      <dt>HB
      <dd>Health and beauty
      <dt>LG
      <dd>Lawn and garden
      <dt>SG
      <dd>Sporting goods
    </dl>
  </info>
  </area>
</help>
```

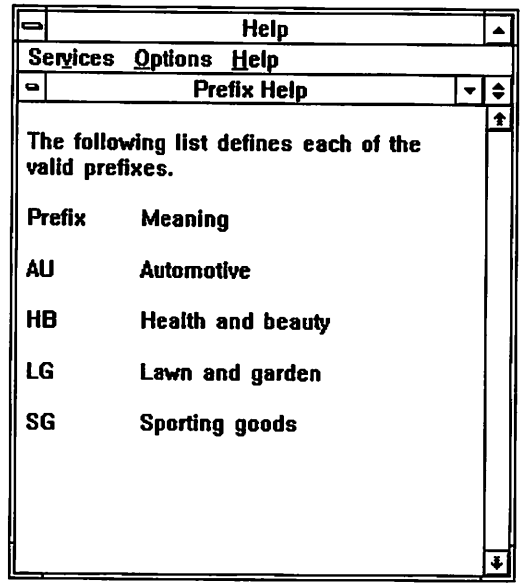
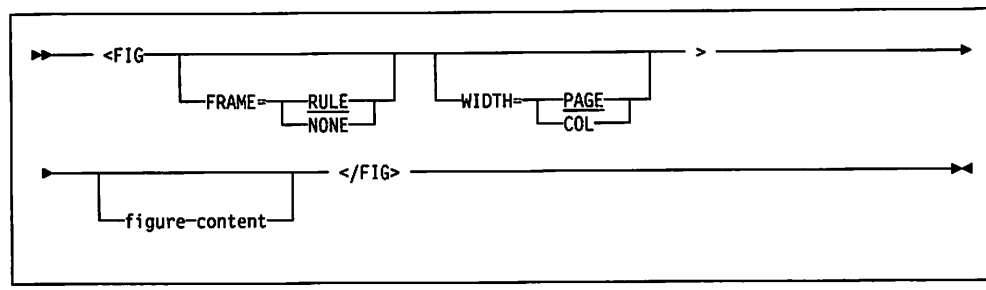


Figure 15-20. Definition Term Header

FIG (Figure)

The FIG tag defines the format of information so that it is set off from the surrounding panel and retains the format of the enclosed text.



FRAME = RULE | NONE
This attribute specifies the type of frame to put around the figure.

RULE Specifies that a line is to be drawn above and below the figure.
NONE Specifies that no frame will appear.

Note: Help panels do not support the FRAME attribute. If you code the FRAME attribute on a help panel, it will be ignored for OS/2 DM applications.

WIDTH = PAGE | COL
This attribute specifies where the figure should be aligned. PAGE will use the original left margin. COL will use the current left margin. The current left margin may be different from the original left margin if another tag (for example, the UL tag) caused text to be indented.

Note: Help panels do not support the WIDTH attribute. If you code the WIDTH attribute on a help panel, it will be ignored for OS/2 DM applications.

figure-content
This specifies the content of the figure definition.

Description

The FIG tag defines the format of information so that it is set off from the surrounding panel and retains the format of the enclosed text. The text is set off from the rest of the panel by a frame or spacing. A figure can also contain a figure caption, identified by the FIGCAP tag (see "FIGCAP (Figure Caption)" on page 15-73).

A FIG tag turns off word-wrapping. Tags that normally cause word-wrapping (such as P, NOTE, or PARML) do not cause word-wrapping when nested within a FIG definition. In addition, blank spaces you code in your text markup are preserved in the figure.

If the text on one line is too long to fit in the width defined by the INFO tag for the information region, the line is truncated to the specified width.

Note: If the interpretation of the text within the FIG tag depends on precise character spacing, you should use a nested XMP tag around the text. This will cause the text to display with a monospaced font.

Conditions

You can use the FIG tag only within an information region.

Within help panels, you cannot nest an XMP tag within a FIG tag.

Nested Tags

You can code the following tags within the FIG tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIGCAP (figure caption)	Application Panel Help Panel	Single	15-73
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
XMP (example)	Application Panel	Multiple	15-202

Example

The figure in the following help panel markup shows the formatted result of a FIG tag. Figure 15-21 shows the formatted result.

```
<help name=fig1>FIG (Figure)
  <area>
  <info>
    <fig frame=none>
      <p>
This figure shows how lines of text are
formatted within the FIG tag.
Even though we have used a
paragraph tag, the text
is not
formatted.
Notice that, just as with the LINES tag,
the input lines in the figure body
are treated as output lines;
they do not run together.
    </fig>
  </info>
  </area>
</help>
```

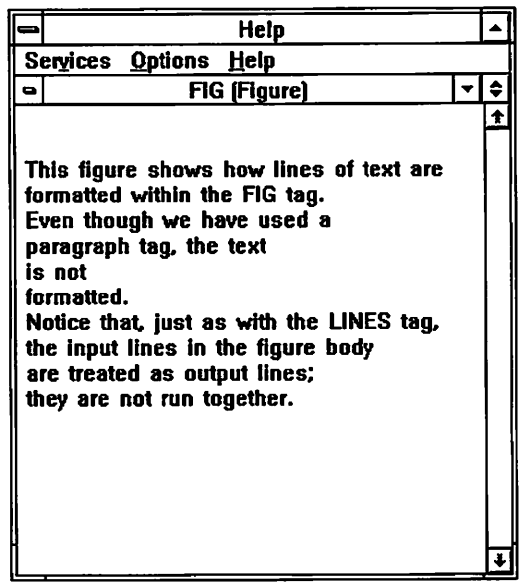


Figure 15-21. Figure

FIGCAP (Figure Caption)

The FIGCAP tag defines a caption for a figure defined with the FIG tag.

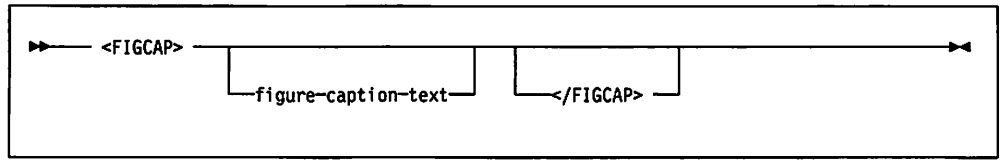


figure-caption-text
This is the text of the figure caption.

Description

The FIGCAP tag defines a caption for a figure defined with the FIG tag. The figure caption appears below the figure unless there is a frame, then the caption appears below the bottom frame. The figure caption is left-aligned.

See "FIG (Figure)" on page 15-70 for more information.

Conditions

You can use the FIGCAP tag only within an information region.

You can code only one FIGCAP tag within a FIG tag and, if used, it must be coded just before the FIG end tag.

Nested Tags

You can code the following tag within the FIGCAP tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Example

The figure in the following panel markup contains a figure caption. Figure 15-22 on page 15-74 shows the formatted result.

```
<panel name=figcap1 width=50 depth=14>Figure with a caption
  <area>
  <info>
    <p>This figure shows the figure caption.
    <fig>

       As you see, the lines of text are
       not formatted.
       The figure caption
       is shown at the bottom
       of the panel.

    <figcap>This is a caption on a figure.
    </fig>
  </info>
  </area>
</panel>
```

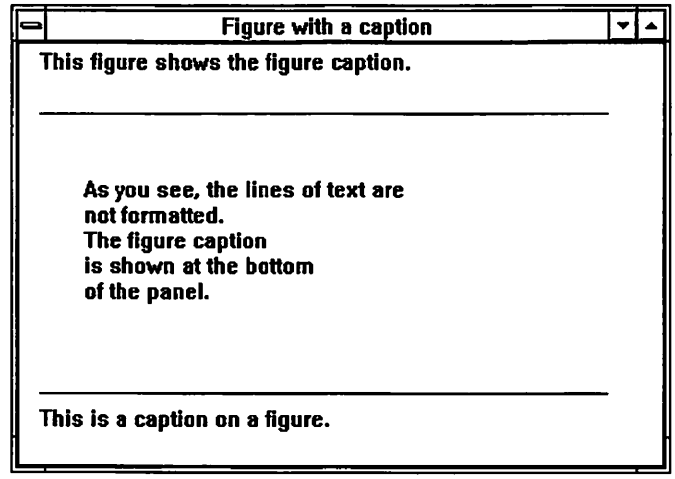
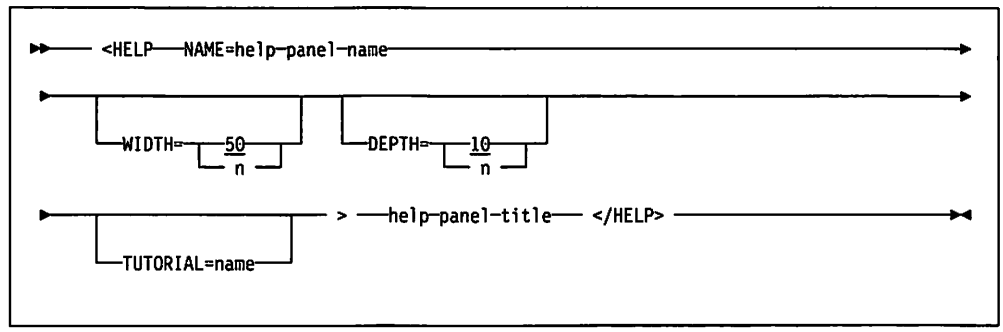


Figure 15-22. Figure Caption

HELP (Help Panel)

The HELP tag defines a help panel.



NAME = help-panel-name
This attribute specifies the name of the help panel.

TUTORIAL = name
This attribute specifies the name of the application-supplied tutorial associated with this panel. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

help-panel-title

Description

The HELP tag defines a help panel.

Conditions

A help panel must contain a scrollable area that you define using the AREA tag.

You must code an INFO tag in a help panel in order for it to have panel content such as paragraphs, headings or notes.

Nested Tags

You can code the following tags within the HELP tag:

Tag	Dialog Element	Usage	Page
AREA (area)	Help Panel	Single	15-11
ICMD (index command)	Help Panel	Multiple	15-79
ISYN (index synonym)	Help Panel	Multiple	15-84
ITOP (index topic)	Help Panel	Multiple	15-86

Note: If you are going to use the ICMD, ISYN and ITOP tags, you must code them prior to the AREA tag.

Example

The following help panel markup contains two paragraphs of text in the defined information region. Figure 15-23 shows the result.

```
<help name=help1>Selecting a Choice
  <area>
  <info>
    <p>After you select an action bar choice, select a choice
    from the action bar pull-down, then press Enter.
    <p>If you decide not to select a choice, press
    the Esc key to exit the panel.
  </info>
  </area>
</help>
```

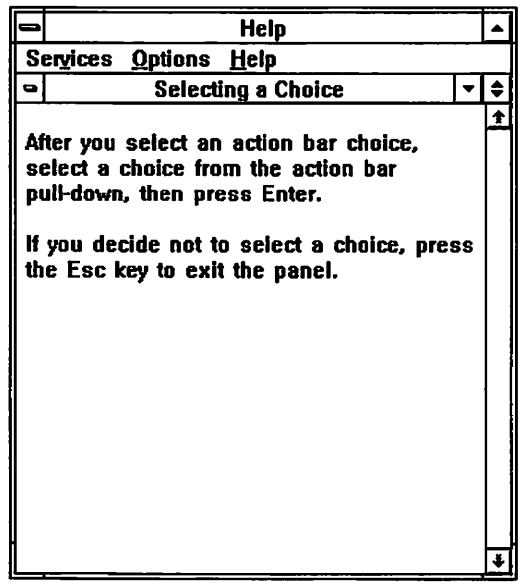
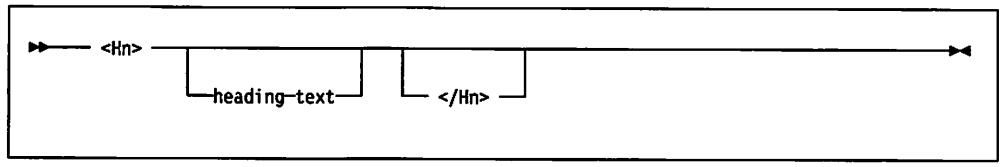


Figure 15-23. Help Panel

Hn (Heading)

The heading tags define main topics and subtopics of information.



heading-text
The *heading-text* specifies the text for the heading.

Description

The heading tags define main topics and subtopics of information. Using these tags, you can define up to four heading levels. The *n* must be a number from 1 through 4.

The heading levels are formatted as follows:

H1
Identifies a main topic of information. The text is centered on the display.

H2, H3, H4
The text is formatted against the left margin of the panel area. Heading levels must be hierarchically sequenced from H1 to H4.

The compiler adds a blank line to the information region before formatting the *heading-text*. If the length of the *heading-text* exceeds the information region width, the text is word-wrapped across multiple lines.

Conditions

You must specify heading levels in sequence. For example, you must specify an H1 before an H2, an H2 before an H3 and an H3 before an H4. You can also specify an H3 following an H4, an H2 following an H3, and an H1 following an H2 within nested headings. Finally, you can code an H1 or an H2 following an H4 (or an H1 following an H3) to begin new headings.

Only the H2 through H4 tags can contain the RP tag and its associated end tag. The RP tag is not allowed on the H1 tag.

You can use the H*n* tag only within an information region.

Nested Tags

You can code the following tag within the H2, H3, H4 tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Example

```
<panel name=hn1 width=50>Department Descriptions
  <area>
  <info>
    <h1>Departments
    <h2>Entertainment
    <p>Our entertainment department carries the finest in home
    entertainment components.
    <h3>Exotic Pets
    <p>You can order from a wide variety of exotic pets and pet
    supplies in this department.
    <h4>Toys
    <p>Your kids will love our wide selection of toys, games, and
    play equipment.
  </info>
  </area>
</panel>
```

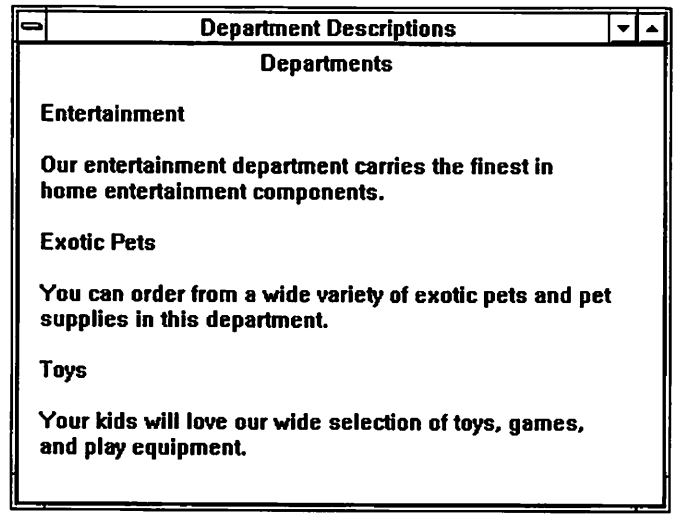
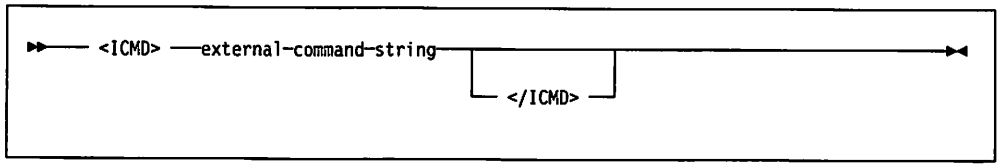


Figure 15-24. Headings

ICMD (Index command)

The ICMD tag designates the help panel that describes a command.



external-command-string
The *external-command-string* specifies the command for which help is being defined. This translatable word must be a minimum of 2 bytes, and cannot exceed 64 bytes. It cannot contain blanks. The text can contain no other tags.

Description

The ICMD tag designates the help panel that describes a command. The help information for the command is assumed to be in the help panel in which the ICMD tag is defined. If the help panel provides help for more than a single command, an ICMD tag should be defined within the HELP tag for each command. The *external-command-string* is used as the index entry in the command index.

The same *external-command-string* (ignoring differences in case) cannot be specified on more than one ICMD tag in a help library; that is, only one help panel can be designated as describing a command. If the same *external-command-string* is found more than once by the compiler (either from the same or different help panels), the duplicate occurrences will be ignored and a warning message will be issued. Only one will be retained.

Index entries for commands do not show up when the user asks to see the help index. If you want these index entries to show up, use the ITOP tag in addition to ICMD tag.

Conditions

This tag must follow a HELP tag or another index tag.

Nested Tags

You can code the following tag within the ICMD tag:

Tag	Dialog Element	Usage	Page
SORTKEY (sort key)	Help Panel	Single	15-167

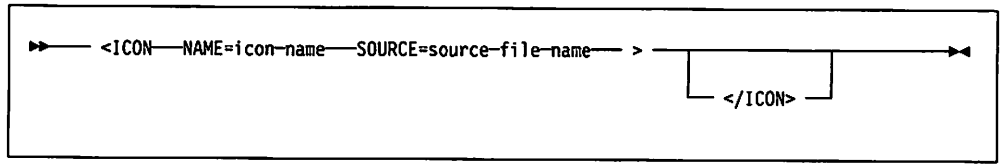
Example

The following help panel markup shows a help panel that contains an index command for the panel. The *external-command-string* of the ICMD definition is the text that appears on the index list panel of commands.

```
<help name=icmd1>Help for Copying
<icmd>Copy
  <area>
  <info>
    <p>You need to specify the following items
    to copy a file.
    <ul>
    <li>File name
    <li>Number of copies
    <p>If you do not specify a number, the
    default is 1.
    <li>Bin number the copy or copies are to be sent to.
  </info>
  </area>
</help>
```

ICON (Icon Resource)

The ICON tag defines an icon resource from a source file that is created by an icon editor.



NAME = icon-name
The name given to this icon resource. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

SOURCE = source-file-name
The name of the file that contains the bitmap of the icon. The compiler creates a dialog element from the information in the source file. You cannot specify a path or extension; the source file extension is always assumed to be .ICO. The file must reside in the current directory or in a directory specified in the system environment variable DPATH.

Description

The ICON tag defines an icon from a source file that is created by an icon editor.

The system variable ZWINICON must be set to the name of the icon so that Dialog Manager can identify the icon displayed when the primary window of this application is minimized.

Conditions

The ICON tag is used only to define the application's minimized icon.

You cannot code this tag within any other tag.

Nested Tags

None.

Example

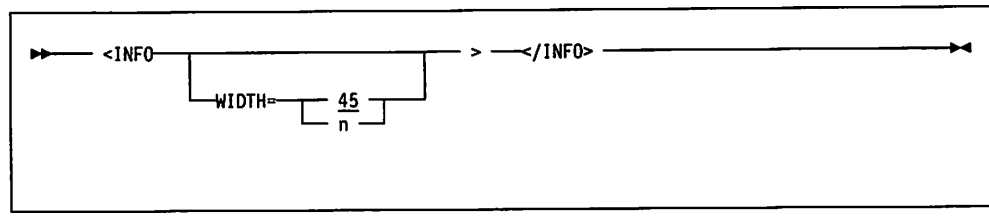
This following markup shows how the ICON tag is used to define the icon resource from the icon bitmap. The file, APP.ICO must contain the icon bitmap, and the system variable, *zwinicon*, must be set to "appicon," the name of the icon. This icon is used when the application's primary window is minimized.

```
<!doctype dm system>

<icon name=appicon source=app>
```

INFO (Information Region)

The INFO tag defines an information region for a panel.



WIDTH = 45 | n

This attribute determines the width, in character units, that the compiler uses to format the text.

The minimum width value is 1 and the maximum is 128.

Note: If you code this attribute on a help panel, it will be ignored.

Description

The INFO tag defines an information region for a panel. The information region is used to display text that can contain paragraphs, notes, examples, figures and several types of lists. Therefore, the text in an information region is static; the user cannot interact with an information region unless it contains a reference phrase.

A detailed description of each tag allowed within the INFO tag appears elsewhere in this chapter.

Conditions

You can use the INFO tag within AREA, PANEL and REGION tags.

Nested Tags

You can code the following tags within the INFO tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel Help Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
Hn (heading)	Application Panel Help Panel	Multiple	15-77
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel Help Panel	Multiple	15-137
SL (simple list)	Application Panel Help Panel	Multiple	15-165

Tag	Dialog Element	Usage	Page
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
XMP (example)	Application Panel Help Panel	Multiple	15-202

Example

The following application panel markup contains an information region with an OL coded within it. Figure 15-25 shows the formatted result.

```
<!doctype dm system>

<panel name=info1 width=80 depth=14>Synonym Choices
  <area>
  <region dir=horiz>
    <divider type=none gutter=1>

    <selfld type=single>Choose a synonym for DABBLE:
      <choice>1. Toy
      <choice>2. Twiddle
      <choice>3. Trifle
      <choice>4. Boondoggle
      <choice>5. Tinker
    </selfld>

    <divider type=none gutter=2>
      <info>
        <ol>
        <li>To amuse oneself idly
        <li>To turn over or around idly or lightly; fiddle with
        <li>To act, perform, or speak, with little seriousness
        <li>To waste time or money on unnecessary work
        <li>To play with machine parts experimentally
        </ol>
      </info>
  </region>
  </area>
</panel>
```

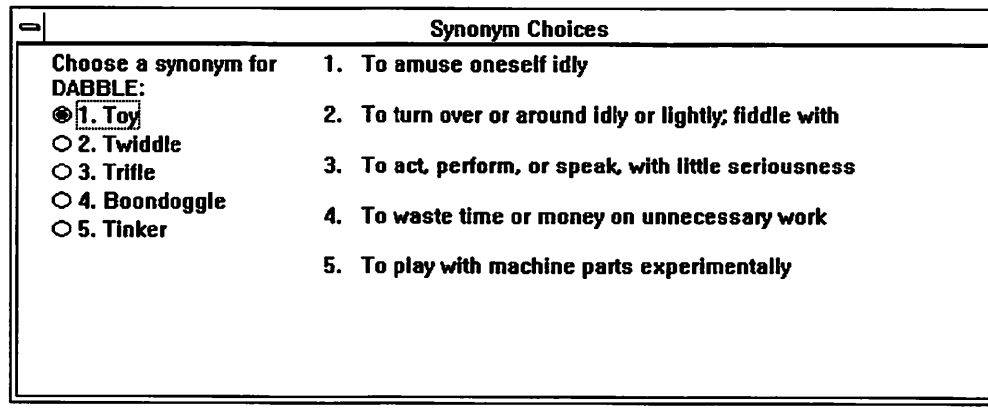
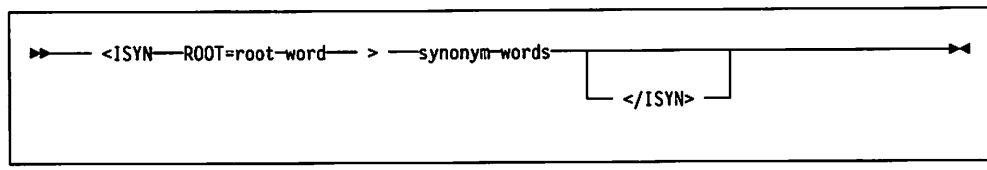


Figure 15-25. Information Region

ISYN (Index synonym)

The ISYN tag defines synonyms for help keywords.



ROOT = root-word
This attribute serves as a link between the various *synonym-words* and their corresponding index topic entries you create with the ITOP tags. You must establish the link by specifying the same *root-word* value in the ROOTS attribute of the appropriate ITOP tags.

The root-word must have the following characteristics:

- 1 – 20 characters in length.
- All characters must be A – Z, a – z, or 0 – 9.

synonym-words
The *synonym words* specify variations and synonyms for the *root-word*. Each *synonym-word* is limited to 40 bytes or less. Use blanks to separate the *synonym words*. For double-byte character sets, either single- or double-byte blanks can be used.

Description

The ISYN tag defines synonyms for help keywords.

These synonyms, which are linked to the ITOP tag through its *root-word-list*, determine the topical entries displayed when the user enters words for a search of the help index. These synonyms are used as search words for the help index. When the help index is searched for any of these variations or synonyms, the user is presented with help topics associated with this ISYN entry through the ITOP.

A help topic defined with an ITOP tag is associated with the synonyms listed in an ISYN tag, if the ISYN *root-word* is in the *root-word-list* of the ITOP tag.

The ISYN tags and their corresponding ITOP tags are used to build an index. You do not need to code an ISYN tag within the help panel to which it relates, but it must be coded in the same source file. The *synonym-words* defined in a help panel can relate to many topics, and therefore to many help panels. Synonyms that you define with this tag do not appear in the visual index.

Conditions

You must code ISYN tags after a HELP tag or another index tag and before an AREA tag.

Nested Tags

None.

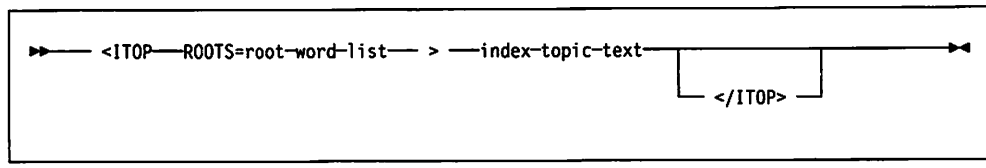
Example

This example defines an index topic for a panel and then defines synonyms for root words. This help panel appears when a user performs a search with the help index and selects the index topic **Copying a Document**.

```
<help name=isyn1>Help for Copying
<isyn root=copy>copy copying duplicate duplicating
<isyn root=folder>folder folders document documents
<itop roots='copy folder'>Copying a Document
  <area>
  <info>
    <p>You need to specify the following items
    to copy a file.
    <ul>
    <li>File name
    <li>Number of copies
    <p>If you do not specify a number, the
    default is 1.
    <li>Bin number the copy or copies should be sent to.
    </ul>
  </info>
  </area>
</help>
```

ITOP (Index Topic)

The ITOP tag defines a help index topic.



ROOTS = root-word-list
The ROOTS attribute specifies a list of root words.

The root-word must have the following characteristics:

- 1 – 20 characters in length.
- All characters must be A – Z, a – z, or 0 – 9.

One or more blanks must separate words, so you must enclose the entire list in quotes.

The *root-word-list* serves as a link between the ITOP tags and the ISYN tags. An ITOP tag is associated with the ISYN tags which use the same *root-words* that appear in the *root-word-list* on the ITOP tag. The user never sees the *root-word-list*. To allow the user to search for an index topic pertaining to a synonym, you must use ISYN tags to map the root-words that are associated with the entry to synonyms in the translated national language.

Note: The *root-word-lists* are not meant to be translated. They serve only to link the ITOP tags and the ISYN tags.

index-topic-text
This is the text of the index entry displayed in the help index list panel when the user selects help index. The *index-topic-text* may contain up to 64 bytes and represents a topic of the help panel in which the ITOP tag is nested. The *index-topic-text* is compared in a case-insensitive manner, based on code page and country information, to eliminate duplicate entries.

Description

The ITOP tag defines a help index topic. It defines the help topic (the text displayed in the help index list panel), the help panel for the topic, and the links to the synonym words relating to the help panel.

Index-topic-text is displayed in the help index list panel when all index topics are requested, or when topics relating to specific synonyms are requested by the user. The topic texts that are displayed for specific synonyms are those that have a root in the ITOP *root-word-list* that matches the ISYN root of the entered synonym words.

The help panel in which the ITOP tag is coded is the panel that will be displayed when the *index-topic-text* is selected during run-time processing of the help index. More than one ITOP tag can be coded within a help panel.

If the same help topic (*index-topic-text*) is defined for more than one help panel (ignoring differences in case) in a file, duplicates will be ignored and a warning message will be issued at compile time. Only the first help topic will be kept. Any duplicates found while building an index at runtime (by merging the open help libraries) will be kept.

Conditions

You must code this tag after a HELP tag or another index tag. .

Nested Tags

You can code the following tag within the ITOP tag:

Tag	Dialog Element	Usage	Page
SORTKEY (sort key)	Help Panel	Single	15-167

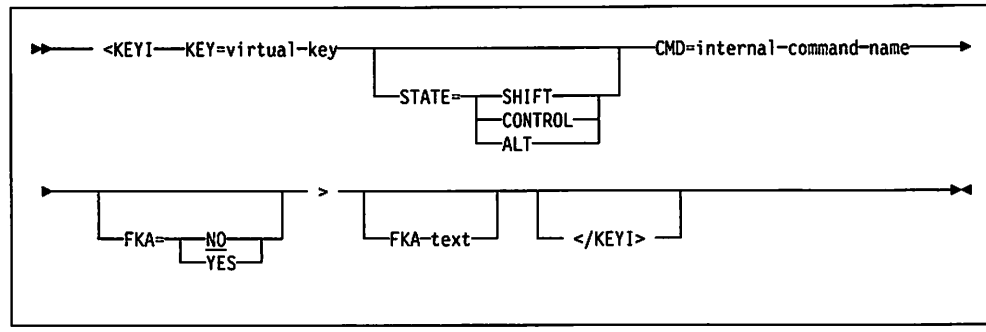
Example

The index topic definition in this help panel specifies the ISYN roots "copy" and "folder." This is the help panel that appears when a user performs a search of the help index for one of the synonyms, then selects the index topic **Copying a document**.

```
<help name=copy03>Help for Copying
<isyn root=copy>copy copying duplicate duplicating
<isyn root=folder>folder folders document documents
<itop roots='copy folder'>Copying a Document
  <area>
  <info>
    <p>You need to specify the following items
    to copy a file.
    <ul>
    <li>File name
    <li>Number of copies
    <p>If you do not specify a number, the
    default is 1.
    <li>Bin number the copy or copies are to be sent to.
    </ul>
  </info>
  </area>
</help>
```

KEYI (Key Item)

The KEYI tag defines a key assignment within a key mapping list.



KEY = virtual-key

This attribute specifies the name of the key to assign to the command.

The following table shows which valid key and shift state combinations are allowed. The virtual key must come from this table. Not all keys are available on all keyboards.

In the table, a "Y" marks a valid combination of keys; a "." marks a combination that is not valid.

Table 15-1. Valid Key and Shift State Combinations

Virtual-key	Alone	SHIFT	ALT	CONTROL
A – Z	.	.	Y	Y
BREAK	Y	.	.	.
ENTER	Y	Y	Y	Y
ESC	Y	.	.	.
F1 – F24	Y	Y	Y	Y
PAUSE	Y	.	.	.
PRINTSCRN	Y	.	.	.

The *virtual-key* is case-insensitive (for example, F3 and f3 are processed the same).

CMD = internal-command-name

The CMD attribute specifies the command that the Dialog Manager issues when the user presses the key. The *internal-command-name* is translated to uppercase.

STATE = shift-state

To assign commands to shifted keys, you supply a STATE attribute with the virtual key.

The valid *shift-states* are SHIFT, CONTROL, and ALT. You cannot assign multiple states to a single virtual-key (for example, STATE = 'SHIFT CONTROL' is not allowed).

Note: Specifying a STATE value implies that a two-key sequence is required for that key item. This means that a key item of Shift+F1 requires the shift to be pressed regardless of the current Caps Lock setting.

FKA = NO | YES
Specifies whether the given key is to appear in the panel's function key area. NO indicates that the key is not to appear.

FKA-text
The *FKA-text* specifies the text for the key which is to appear in the function key area of panels on which this key mapping list is used. If not specified, the *FKA-text* will default to the external command name (if any). If there is none, it will default to the *internal-command-name*.

Key engravings will be added to function keys F1 – F24 only. When constructing the function key area, the Dialog Manager concatenates a string consisting of the key engraving followed by an equal sign in front of the specified *FKA-text* for function keys F1 – F24. However, if the *FKA-text* is the same as the key engraving, ignoring differences in case, only the FKA-text is used. (The key engraving is omitted.) The key engravings for keys other than F1 – F24 will not be displayed.

The function key area is formatted at run time based on the panel size.

If the *FKA-text* exceeds 64 bytes, the compiler truncates the text and issues a warning message. The Dialog Manager will format the text to fit within the corresponding window. This could require further truncation.

Description

The KEYI tag defines a key assignment within a key mapping list. If the compiler finds a KEYI tag with the same valid key and shift state combination as a previous key in the key mapping list, it replaces the previous definition with the new one. This processing allows embedding of an entity containing common key definitions before the first KEYI tag within a key mapping list. KEYI definitions specific to the key mapping list will replace any shared definitions they duplicate.

The compiler automatically provides each key mapping list with default definitions for Enter (ENTER command), Esc (CANCEL command), F1 (HELP command), F3 (EXIT command), and F10 (ACTIONS command). Since FKA = NO is the default, none will be displayed in the function key area. These key definitions are stored in your key mapping list unless you replace them with your own KEYI definitions.

When the Dialog Manager processes a key mapping list for display, it places the key assignments in the function key area in the following order, regardless of the order they are defined in the key mapping list:

1. Enter
2. Esc
3. F1...F24
4. Shift+F1...Shift+F24
5. Alt+F1...Alt+F24
6. Ctrl+F1...Ctrl+F24
7. Break
8. Pause
9. Print Screen
10. Shift+Enter
11. Alt+A...Alt+Z
12. Alt+Enter
13. Ctrl+A...Ctrl+Z
14. Ctrl+Enter.

Conditions

You must code the KEYI tag within a KEYL tag.

Each key can only have one command assigned to it.

The ACTIONS command can only be mapped to the F10 key. If assigned to any other key or if the STATE attribute is also specified, the compiler will issue a warning message and the key assignment will be ignored. If assigned the ACTIONS command, F10 will not appear in the function key area, regardless of the value of the FKA attribute of the KEYI tag.

F1 can only be defined to issue the HELP command. If mapped to any other command, a warning message will be issued and the key assignment will be ignored.

Nested Tags

None.

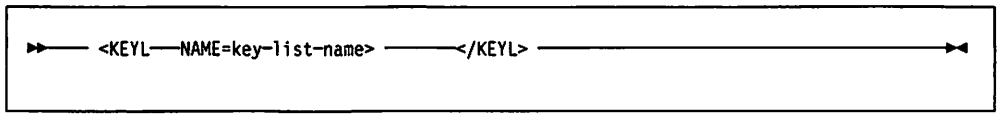
Example

The following markup contains several key mapping assignments. This key list will be displayed in the function key area of panels referencing the *mainkeys* key list.

```
<!doctype dm system>
⋮
<keyl name=mainkeys>
  <keyi key=F7    fka=yes cmd=BACKWARD >Backward
  <keyi key=F8    fka=yes cmd=FORWARD  >Forward
</keyl>
⋮
```

KEYL (Key Mapping List)

The KEYL tag defines a key mapping list where keys are mapped to commands.



NAME = key-list-name

The NAME attribute specifies a name for a key mapping list. The PANEL and PANDEF tag reference the *key-list-name*. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

Description

The KEYL tag defines a key mapping list where keys are mapped to commands.

Conditions

The KEYL tag cannot be nested within any other tags.

Nested Tags

You can code the following tag within the KEYL tag:

Tag	Dialog Element	Usage	Page
KEYI (key item)	Key Mapping List	Multiple	15-88

Example

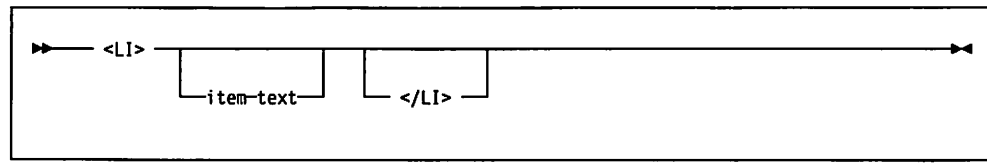
The following markup shows a key mapping list for a typical application panel. The commands FORWARD and BACKWARD are mapped to function keys. The name of the key mapping list, *mainkeys*, should be referenced on the KEYLIST attribute of the PANEL or PANDEF tag in order for these keys to be active when the panel is displayed.

```
<!doctype dm system>
:
<keyl name=mainkeys>

  <keyi key=F7    cmd=BACKWARD >Backward
  <keyi key=F8    cmd=FORWARD  >Forward
</keyl>
:
```

LI (List Item)

The LI tag defines a list item within an ordered list, unordered list, or simple list.



item-text
Specifies the text of the list item.

Description

The LI tag defines a list item within an ordered list, unordered list, or simple list.

The formatting of the LI tag depends on the type of list you use it within and the level of nesting.

List	Formatting
Ordered	Formats with a 4-space indentation within the level of the list in which it is defined and is preceded by sequential numbers or letters.
Unordered	Formats with a 4-space indentation within the level of the list in which it is defined and is preceded by bullets or dashes.
Simple	Formats with a 4-space indentation within the level of list in which it is defined.

The next list item implicitly ends the previous list item.

If *item-text* is not specified, a blank line is displayed for that item.

Conditions

You can use the LI tag only within an information region.

The LI tag can only be coded within the OL, UL, or SL tags.

Nested Tags

You can code the following tags within an LI tag:

Tag	Dialog Element	Usage	Page
CAUTION (caution)	Application Panel Help Panel	Single	15-20
DL (definition list)	Application Panel Help Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121

Tag	Dialog Element	Usage	Page
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel Help Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
WARNING (warning)	Application Panel Help Panel	Single	15-192
XMP (example)	Application Panel Help Panel	Multiple	15-202

Example

The following help panel markup contains a unordered list with four list items. Figure 15-26 shows the formatted result.

```
<help name=li1>Help for the Fifth Floor
  <area>
  <info>
    <p>The following departments are located on the fifth floor:
    <ul>
      <li>Electronics
      <li>Household goods
      <p>Excluding major appliances
      <li>Hardware
      <li>Records and tapes
    </ul>
  </info>
  </area>
</help>
```

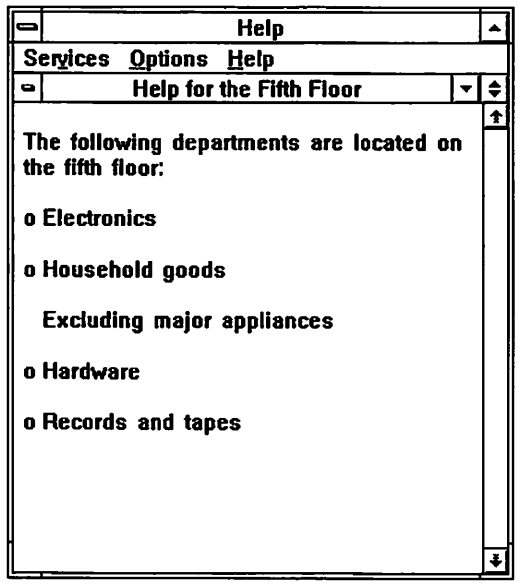
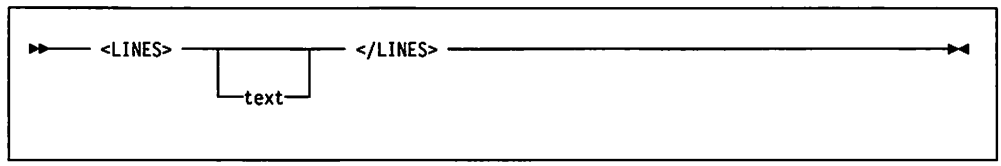


Figure 15-26. List Items

LINES (Lines)

The LINES tag defines unformatted text to be displayed at run time.



text
This specifies the unformatted text.

Description

The LINES tag defines unformatted text to be displayed at run time. That is, the compiler will end each line at the same point where you ended the line in the source file, and the text will display without being formatted into paragraphs. If the source text on one line is too long to fit the width defined by the INFO tag for the information region, the line is truncated to the width specified on the INFO tag. Tags that normally cause word-wrapping (such as the P, LI, or CAUTION) do not cause word-wrapping when embedded within the LINES tag.

If the source text on one line is too long to fit on the panel with the width defined, the line is truncated.

The formatting of the LINES tag is similar to that of the FIG tag, except that there is no border or caption capability.

Note: If the interpretation of the text within the LINES tag depends on precise character spacing, you should use a nested XMP tag around the text. This will cause the text to display with a monospaced font.

Conditions

The LINES tag is valid only within an information region.

Within help panels, you cannot nest an XMP tag within a LINES tag.

Nested Tags

You can code the following tags within the LINES tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165

Tag	Dialog Element	Usage	Page
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
XMP (example)	Application Panel	Multiple	15-202

Example

In the following markup, the lines tag is used to show how information randomly placed in the panel is presented. Figure 15-27 shows the result.

```
<help name=lines1>Special Activities
  <area>
  <info>
    <lines>
    This example of the LINES tag shows
    how the text      f l o w s...

            Notice how the text is placed
            just as it is coded in the source file.

                              The   text   is   not
                      formatted.

    Did this format the way you expected ?
    </lines>
    <p>Notice how the paragraph tag formats the text.
    The

    blank spaces between
    the lines are ignored and the text is formatted to flow in the output file.
  </info>
  </area>
</help>
```

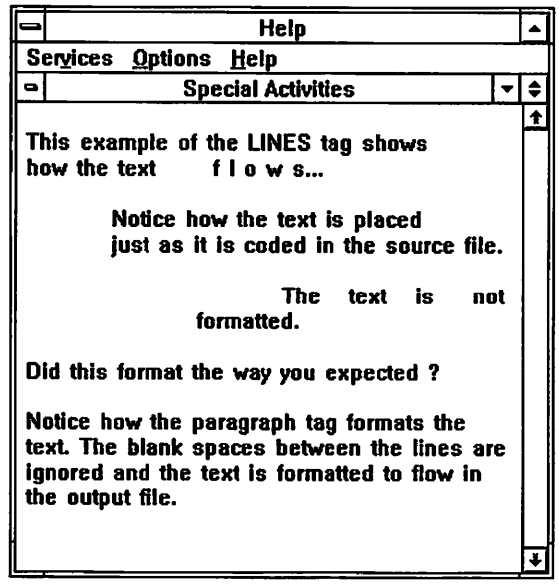
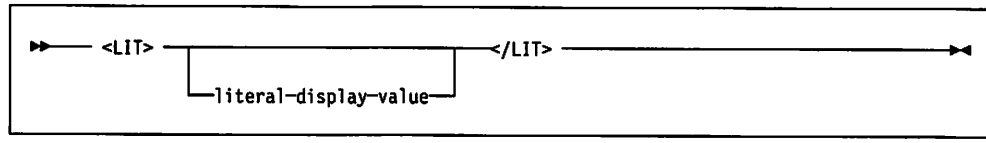


Figure 15-27. Lines

LIT (Literal)

The LIT tag defines a string where all blanks are significant and included in the value.



literal-display-value
This defines a string with all blanks preserved.

Description

The LIT tag defines a string where all blanks are significant and included in the value. No stripping of leading, trailing, or embedded blanks is performed.

This is the only way to specify trailing blanks or a value of all blanks in the XLATI *displayed-value*.

The </LIT> tag should be on the same line as the *literal-display-value* to preserve the original spacing of the value and to indicate where the trailing spaces end.

Conditions

This tag is valid only within an XLATI tag.

Multiple LIT tags can be coded, as long as they are not nested within each other.

Nested Tags

None.

Example

The following markup shows how to use the LIT tag to preserve trailing blanks or to code a string of all blanks.

```
<!doctype dm system>

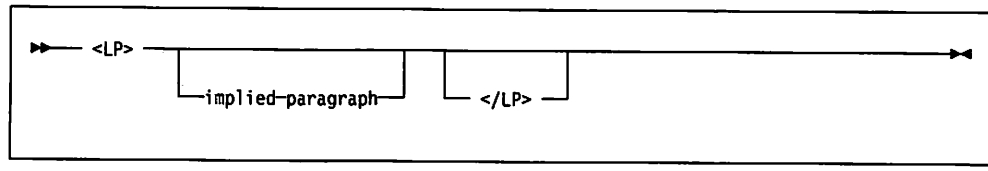
<varclass name=vstring type='CHAR 1' >
   <xlatl format=none>
      <xlati value=0><lit> </lit>
      <xlati value=2>trailing blanks<lit>        </lit>
   </xlatl>
</varclass>

<varlist>
   <vardcl name=lstring varclass=vstring>
</varlist>

<panel name=lit01>
   <area>
     <dtacol>
        <dtafld datavar=lstring>Type a string:
     </dtacol>
   </area>
</panel>
```

LP (List Part)

The LP tag defines a comment or explanation within an ordered list, unordered list, or simple list.



Implied-paragraph
This indicates the text of the implied paragraph.

Description

The LP tag defines a comment or explanation within an ordered list, unordered list, or simple list. You can code the LP tag anywhere within a list.

The text following the LP tag starts at the left margin of the current level of the list. It is not numbered or lettered. When you use it with the OL tag, LP does not interrupt or increment the sequence.

The next list item or the end of the list implicitly ends the list part.

Conditions

You can use the LP tag only within an information region.

You can use the LP tag only within the OL, UL, or SL tags.

Nested Tags

You can code the following tags within the LP tag:

Tag	Dialog Element	Usage	Page
CAUTION (caution)	Application Panel Help Panel	Single	15-20
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (parameter term)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165

Tag	Dialog Element	Usage	Page
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
WARNING (warning)	Application Panel Help Panel	Single	15-192
XMP (example)	Application Panel Help Panel	Multiple	15-202

Example

The following help panel markup contains an unordered list. A list part definition is nested within the third list item of the unordered list. Figure 15-28 shows the formatted result.

```
<panel name=lp1>Sales
  <area>
  <info>
    <p>The following departments are offering discount
    sales on all items in stock throughout the month
    of August.
    <ul>
      <li>Appliances
      <li>Automotive
      <li>Hardware
      <lp>Items are extremely limited in this department, so hurry!
      <li>Sporting goods
    </ul>
  </info>
  </area>
</panel>
```

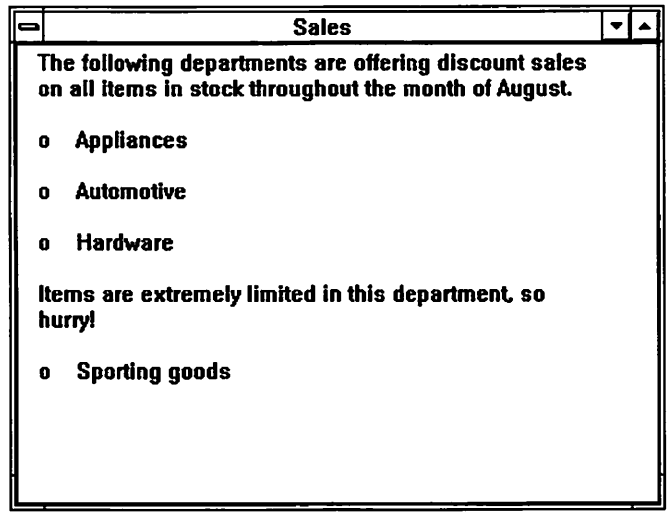
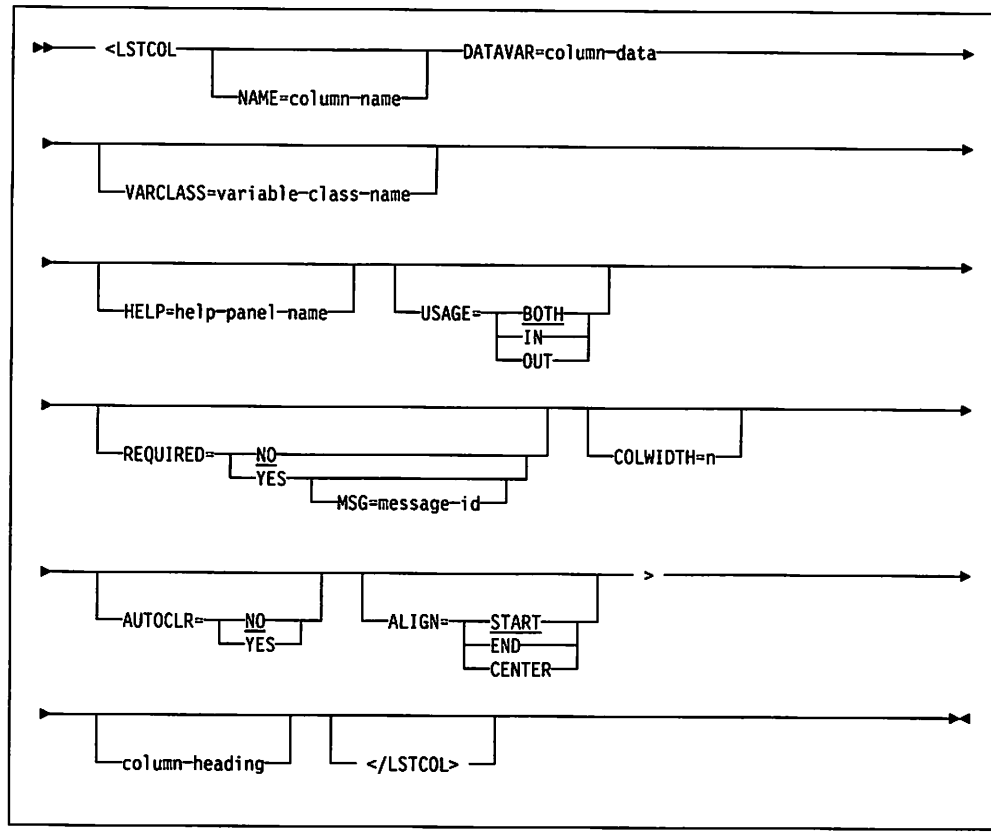


Figure 15-28. List Part

LSTCOL (List Column)

The LSTCOL tag defines a column within a list field.



NAME = column-name

Specifies the name of the list column.

This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the list field.
- On the ADDPOP call to position a pop-up window relative to the list field.
- On the DISPLAY call to place the cursor in the list column.
- On the PANEL definition to place the cursor in the list column.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

DATAVAR = column-data

This is the data that will occupy the column. The variable named must be an array declared with the VARDCL tag. Do not use % notation.

All array variables referenced within a single list column must have a maximum dimension (declared within a VARDCL tag) that is greater than or equal to the NUMROWS attribute of LSTFLD.

Note: If the dimension of all the arrays is not greater than or equal to the NUMROWS attribute, a warning is issued and the NUMROWS attribute is set to the smallest array dimension.

The maximum length of a data value that can be displayed or entered (with possible scrolling) within a LSTCOL entry is determined by the characteristics of:

- The explicitly specified VARCLASS, if any
- The default VARCLASS for the DATAVAR.

Any array variable named in a LSTCOL must be defined explicitly to the Dialog Manager before issuing a DISPLAY service call to display the panel.

VARCLASS = variable-class-name
This attribute specifies the name of the variable class defined using the VARCLASS tag. The variable class overrides the default variable class referenced by the VARDCL that declared the data variable for this column. The TYPE information of the specified variable class must be the same as the default VARCLASS for DATAVAR.

If the VARCLASS specified does not have TYPE information that matches the type information of the overridden VARCLASS, a warning message is issued .

HELP = help-panel-name
This attribute specifies the name of a help panel for this column.

If the user selects help on a list column, the help defined for the list column displays. If no help is specified for the list column, the help inherits the help specified on the LSTFLD tag. If no help was defined for LSTCOL or LSTFLD, the extended help panel displays. If an extended help panel was not defined, a message displays stating that no help is defined. This attribute is valid only when USAGE = IN or BOTH. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

USAGE = BOTH | IN | OUT
This attribute indicates if this column is for input, output, or both.

REQUIRED = NO | YES
This indicates if this column is required to have input for any modified record (that is, the REQUIRED = YES parameter is only enforced for those rows that are modified). The default is NO, input is not required. This is valid only when USAGE = IN or BOTH. It is ignored if you specify USAGE = OUT.

MSG = message-id
This attribute is only valid when REQUIRED = YES. It specifies the message to be displayed when a required entry is not made.

Note: You can specify messages pertaining to other validations using the XLATL and CHECKL tags within a VARCLASS. See "CHECKL (Validity Check List)" on page 15-26 and "XLATL (Translate List)" on page 15-197 for more information.

COLWIDTH = n

The COLWIDTH attribute serves as a boundary to the column width. The justification of the data within the column is taken from the ALIGN attribute on the LSTCOL tag. If the data size exceeds the column size, the behavior depends on whether the column can accept input or not. The range is 1 to 128.

- For input/both type columns, the data is made scrollable up to the number of characters taken from the associated VARCLASS.
- For output-type columns, the variable data is truncated at the column width.

If a COLWIDTH is not specified, its default depends on whether the column can take input or not.

- For input/both type columns, the column width is set to the data size or 25, whichever is less, and the above rules apply.
- For output-type columns, the column width is the minimum of 25 or the length of the data.

Note: Some adjustments may be made to the column size to avoid truncating the column heading text. This adjustment is made after the determination of the data size. For output-only columns, this means more data may be displayed than the COLWIDTH attribute indicates.

AUTOCLR = NO | YES

This attribute specifies whether the field is to be an auto-clear field. When the first character is typed in an auto-clear field, the rest of the field is cleared automatically. This attribute is valid only when USAGE = IN or BOTH.

ALIGN = START | END | CENTER

The ALIGN attribute specifies how the data value is to be displayed in the list field.

If you specify the START alignment, the value is left-justified in the field. If you specify the END alignment, the value is right-justified and if you specify the CENTER alignment, the value is centered.

Output fields are not scrollable; however, input fields may be scrollable as necessary.

column-heading

The text used for the title of the column. The column heading is centered above each column independent of the value of the ALIGN attribute. If the column-heading width and the COLWIDTH values are not equal, the greater of the two is used. If the column heading and the COLWIDTH attributes are omitted, the size of the column variable determines the width.

Description

The LSTCOL tag defines a column within a list field. Multiple columns can be defined.

If the list field is scrollable, all columns defined within that list field are scrollable. Multiple columns within a list field scroll together and cannot be scrolled independently of each other.

Conditions

The LSTCOL tag can only be coded within a LSTFLD tag.

The column headings do not scroll vertically.

Nested Tags

None.

Example

This example shows vertical rules in a list field, and center and right aligned list columns. Figure 15-29 on page 15-105 shows the formatted result.

```
<!doctype dm system>

<varclass name=vc1     type ='char 20'>
<varclass name=vc2     type ='char 7'>
<varclass name=qtycls  type ='char 7'>
<varclass name=priccls type ='numeric 3 2'>
<varclass name=daycls  type = idate>

<varlist>
  <vardcl name=cname    varclass=vc1>
  <vardcl name=cnum     varclass=vc2>
  <vardcl name=itemcol  varclass=vc2     maxdim=500>
  <vardcl name=qtycol   varclass=qtycls  maxdim=500>
  <vardcl name=pricecol varclass=priccls maxdim=500>
  <vardcl name=zidate   varclass=daycls>
</varlist>

<panel name=lstfld2>Wholesale Food Distributors
  <ab>

:
  </ab>
 <topinst>Here are the items currently ordered by this customer.
 <area>
 <dtafld datavar=cname usage=out>Customer name:
 <dtafld datavar=cnum  usage=out>Customer number:
 <lstfld rules=vert>
   <lstcol datavar=itemcol>Item
   <lstcol datavar=qtycol align=center>Quantity
   <lstcol datavar=pricecol align=end>Price
 </lstfld>
 <dtafld datavar=zidate usage=out>Delivery date:
 </area>
</panel>
```

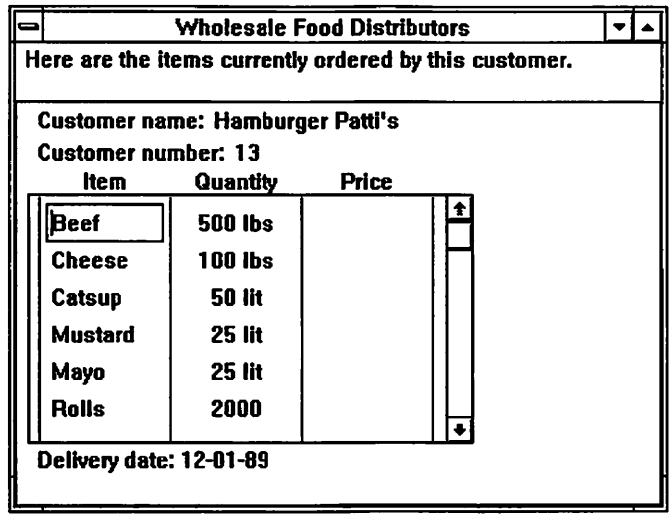
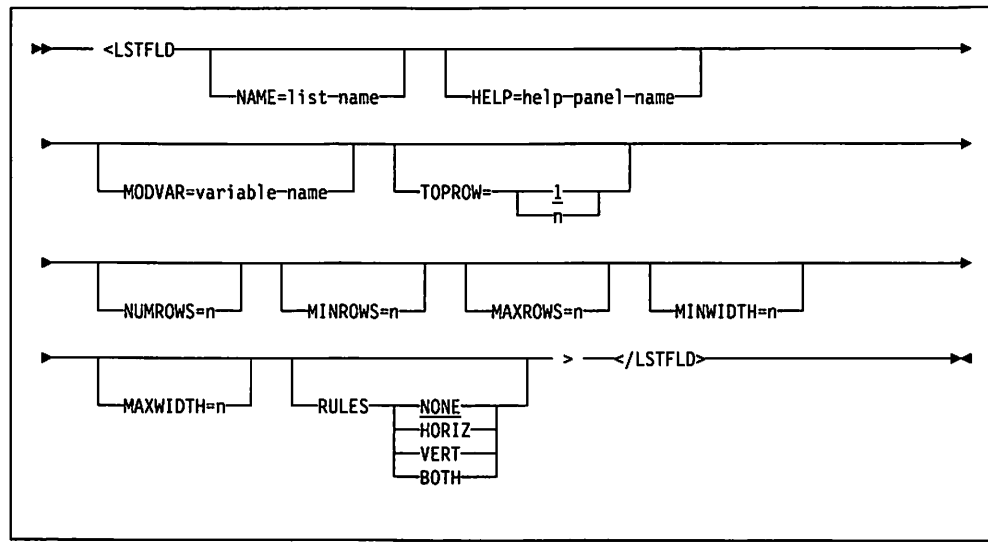


Figure 15-29. List Columns

LSTFLD (List Field)

The LSTFLD tag defines a scrollable list that is made up of columns of data from array variables.



NAME = list-name

Specifies the name of the list field.

This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the list field.
- On the ADDPOP call to position a pop-up window relative to the list field.
- On the DISPLAY call to place the cursor in the list field.
- On the PANEL definition to place the cursor in the list field.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

This attribute specifies the help panel for a list field. This help can be superseded by the help specified on the LSTCOL tag. If no help was defined for LSTCOL or LSTFLD, the extended help panel is displayed. If an extended help panel was not defined, a message is displayed stating that no help is defined. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

MODVAR = variable-name

This attribute specifies the name of the array variable that indicates which rows in the list field have been modified. Do not use % notation.

Normally, an application will initialize the first element of the array to 0 and then call the DISPLAY service. After returning from the DISPLAY service call, the non-zero elements before the first 0 entry each contain the row number of a modified row (the first row is row number 1). The sequence of modified-row numbers ends with the first 0 element or the end of the array, whichever occurs first.

The dimension of the MODVAR array determines the total number of items the Dialog Manager can return as modified.

An application may initialize the array to a sequence of row numbers (ended by a 0 element or the end of the array) before the DISPLAY service call to indicate that a group of rows has been premodified. Combined with the REQUIRED=YES attribute on the LSTCOL tag, this enables an application to enforce the required input attribute for a list field row.

TOPROW=1 | n

The TOPROW attribute determines which element in the array or row in the table is to appear at the top of the list column when first displayed. The default is 1. The value specified on the TOPROWS attribute cannot be greater than the value for NUMROWS. This attribute supports variable substitution with % notation, so the application may set it at run time. (For example, to specify the variable *vtoprow*, code '%vtoprow'.)

NUMROWS=n

The NUMROWS attribute determines the number of rows or data elements in the list. This attribute supports variable substitution with % notation. (For example, to specify the variable *vnumrow*, code '%vnumrow'.) This determines the range over which the entire list field scrolls. If specified, this value must be less than or equal to the maximum dimension of each array variable referenced by the list field. If not specified, the smallest maximum dimension of these array variables is taken to be the limit.

Note: If the NUMROWS specified is greater than the maximum dimension of an array variable referenced, a warning is issued and the NUMROWS is set to the smallest maximum dimension of the array variables.

If this condition is detected at run time (that is, if NUMROWS identifies a dialog variable), the DISPLAY service will end and a return code indicating the error will be provided.

The range is 1—1024 rows and the default is set by the actual array dimensions.

MINROWS=n

MINROWS specifies the minimum number of rows of the list field that can be visible on the panel. The value specified on this attribute must be less than or equal to the value for MAXROWS. The default is 1. The range is 0—512 rows.

Note: If this attribute is greater than the value specified in NUMROWS, the NUMROWS value is used. This evaluation is made at run time, because NUMROWS can be a variable.

MAXROWS=n

MAXROWS specifies the maximum number of rows of the list field that can be visible on the panel. If this value is not specified, the value specified in NUMROWS is used if it is less than 32. Otherwise, MAXROWS is set to 32 or the value specified on the MINROWS attribute, whichever is greater. The range is 1—512 rows.

Note: If this attribute is greater than the value specified in NUMROWS, the NUMROWS value is used. This evaluation is made at run time, because NUMROWS can be a variable.

MINWIDTH = n
The minimum width of the list field, specified in character units. The default is 8. The range is 1 – 512.

MAXWIDTH = n
The maximum width of the list field, specified in character units. If this value is not specified, the list field will take up as much extra width as is available on the panel, until all columns of data are displayed. The default is 64. The range of 1 – 512.

RULES = NONE | HORIZ | VERT | BOTH
The RULES attribute specifies the type of interior rules that will be drawn in the list being defined with the LSTFLD tag. This applies to all the list columns within the context of this tag.

Description

The LSTFLD tag defines a scrollable list that is made up of columns of data from array variables.

If you want a fixed width always visible for your list field, specify the same value for MINWIDTH and MAXWIDTH. For a fixed number of rows to always be visible, you must specify the same value for MINROWS and MAXROWS.

If portions of the list are not visible upon display, list fields are scrollable.

Conditions

You must code a LSTCOL tag within this tag.

Nested Tags

You can code the following tag within the LSTFLD tag:

Tag	Dialog Element	Usage	Page
LSTCOL (list column)	Application Panel	Multiple	15-101

Example

This example shows vertical rules in a list field, and center and right aligned list columns. Figure 15-30 on page 15-109 shows the formatted result.

```
<!doctype dm system>

<varclass name=vc1     type ='char 20'>
<varclass name=vc2     type ='char 7'>
<varclass name=qtycls  type ='char 7'>
<varclass name=priccls type ='numeric 3 2'>
<varclass name=daycls  type = idate>

<varlist>
  <vardcl name=cname    varclass=vc1>
  <vardcl name=cnum     varclass=vc2>
  <vardcl name=itemcol  varclass=vc2     maxdim=500>
  <vardcl name=qtycol   varclass=qtycls  maxdim=500>
  <vardcl name=pricecol varclass=priccls maxdim=500>
  <vardcl name=zidate   varclass=daycls>
</varlist>

<panel name=lstfld2>Wholesale Food Distributors
  <ab>

:
  </ab>
 <topinst>Here are the items currently ordered by this customer.
 <area>
 <dtafld datavar=cname usage=out>Customer name:
 <dtafld datavar=cnum  usage=out>Customer number:
 <lstfld rules=vert>
   <lstcol datavar=itemcol>Item
   <lstcol datavar=qtycol align=center>Quantity
   <lstcol datavar=pricecol align=end>Price
 </lstfld>
 <dtafld datavar=zidate usage=out>Delivery date:
 </area>
</panel>
```

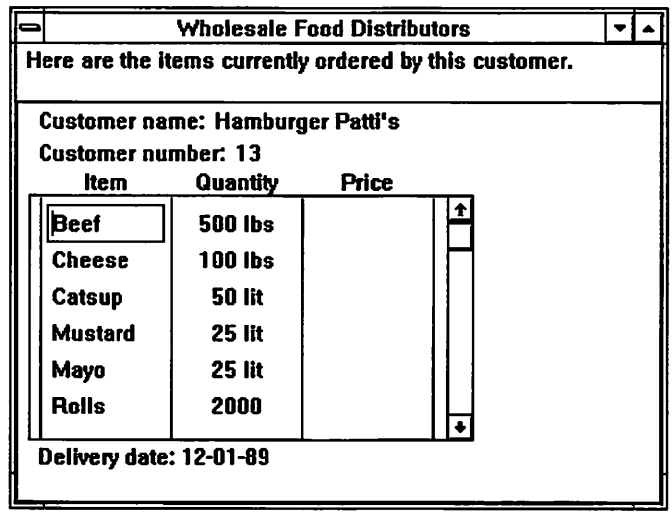
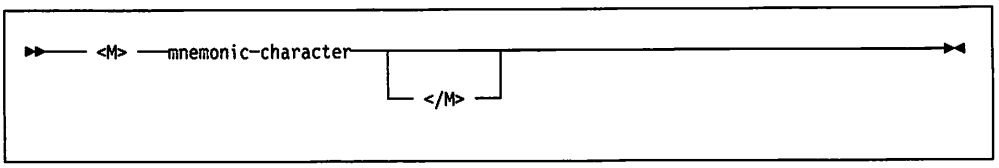


Figure 15-30. List Field

M (Mnemonic)

The M tag defines a mnemonic for a choice.



mnemonic-character

The single-byte character following the mnemonic start tag specifies the mnemonic for the selection field choice, action bar choice or pull-down choice. The mnemonic character must be a single-byte character, double-byte characters are not allowed. In addition, you cannot specify the space character or any of the ASCII control characters as mnemonics:

- ASCII code 0 to 31 (decimal) inclusive
- ASCII code 127 (decimal)
- ASCII code 255 (decimal).

Note: The Presentation Manager uses a tilde (ASCII code 126 decimal) in the text of the choice to indicate the mnemonic character. Since the Presentation Manager's use of the tilde conflicts with the mnemonic tag, you cannot specify a tilde in choice text prior to the mnemonic tag. A tilde may be specified as the mnemonic character, or anywhere in the choice text following the mnemonic character.

If you want the mnemonic to be a character that is not part of the normal choice text, follow the choice text with the mnemonic character specified within the parentheses. This convention is particularly useful when you have a large number of items, which make it difficult to choose a unique mnemonic for each pull-down choice.

For example, if you had the pull-down choice **Add**, you might choose the "A" or the "d" as the mnemonic character. If these mnemonics were already used in the selection field, action bar, or pull-down, you might choose another character, say "B," as the mnemonic and code the following in your tag source:

```
<pdc>Add(<m>B)
```

Description

The M tag defines a mnemonic for a choice. A mnemonic character is underlined to indicate to the user that the mnemonic selection is available. Code the M tag in the text that describes the choice.

Mnemonics provide keyboard users a fast-path technique for selecting action bar choices and pull-down choices from an action bar, or choices within a selection field. When the user enters a valid mnemonic, the action bar choice, pull-down choice or selection field choice containing that mnemonic is automatically selected and the cursor is moved to the selected choice. If the mnemonic entered belongs to an unavailable choice, Dialog Manager returns an audible warning.

The scope of the mnemonic is limited to where the input focus is on the panel. For example, if the input focus is on the action bar with a pull-down menu displayed and a valid mnemonic is entered for a pull-down choice, that choice is automatically selected. If the input focus is on the action bar, with no pull-down

menus displayed, and a valid mnemonic is entered for a pull-down choice, that choice is not selected.

Conditions

The use of mnemonics must be consistent for each action bar choice (ABC tag) and pull-down choice (PDC tag) for an action bar, and CHOICE definitions for a selection field. That is, all ABC, PDC, or CHOICE items nested within AB, ABC, or SELFLD must have unique mnemonics or must have none at all.

The M tag must immediately precede the character in the descriptive text of an ABC tag, PDC tag, or selection field item that is the mnemonic.

You should not specify a single-byte character set Katakana character as a mnemonic.

Nested Tags

None.

Example

The following markup shows how the M (mnemonic) tag is used on an action bar choice, pull-down choice, single-choice selection field, and multiple-choice selection field. Each of the action bar choices and each of the selection field choices in the markup contain an M tag that defines the first character following the mnemonic start tag as the mnemonic for the choice. Figure 15-31 on page 15-113 shows the formatted result.

```
<!doctype dm system>

<panel name=m1 depth=16 width=60>Automobile Service
  <ab>
    <abc><m>File
      <pdc><m>New
        <action run=New>
      <pdc><m>Open...
        <action run=Open>
      <pdsep>
      <pdc><m>Save
        <action run=Save>
      <pdc>Save <m>as...
        <action run=SaveAs>
      <pdsep>
      <pdc>E<m>xit
        <action run=Exit>
    </abc>

    <abc><m>View

    <abc><m>Options

    <abc><m>Help

  </ab>
  <topinst>Choose the most convenient day for your appointment
           and the services needed.
  <region dir=horiz>
    <divider>
    <selfld>Weekdays:
      <choice><m>Monday
      <choice><m>Tuesday
      <choice><m>Wednesday
      <choice>T<m>hursday
      <choice><m>Friday
    </selfld>
    <divider>
    <selfld type=multi>Services available:
      <choice><m>Oil change and filter check
      <choice><m>Tires rotated
      <choice><m>Front end alignment
      <choice>Full <m>service tune-up
      <choice><m>Replace brake pads
      <choice><m>Inspection sticker
    </selfld>
  </region>
</panel>
```

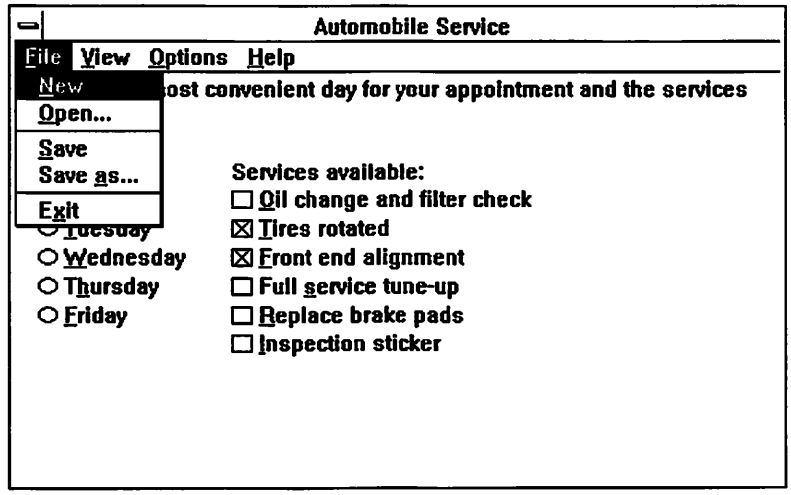
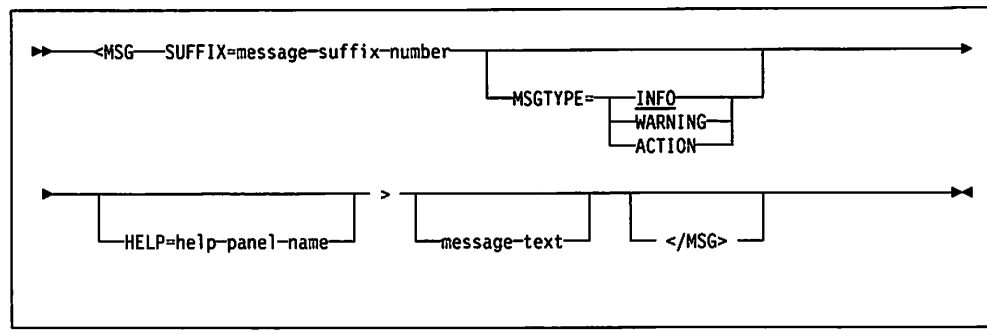


Figure 15-31. Mnemonics

MSG (Message)

The MSG tag defines messages within a message member.



SUFFIX = message-suffix-number
This attribute specifies the suffix of the message. The suffix consists of one numeric character (0 – 9) and is added to the message member name to form the message identifier. The suffix must be unique within a message member. If a duplicate suffix is specified within a MSGMBR, the compiler issues an error message.

MSGTYPE = INFO | WARNING | ACTION
The MSGTYPE attribute specifies the severity of the message. All messages are displayed in a message pop-up. The Dialog Manager displays INFO messages without an alarm and WARNING and ACTION messages with an audible alarm.

Note: You can use the word ERROR as a substitute for ACTION in the MSGTYPE attribute.

HELP = help-panel-name
This attribute specifies the name of the help panel associated with this message that will be displayed if the user selects help for the message. The help panel is defined using the HELP tag. If the attribute is not specified, no help is available for this message.

message-text
Message-text specifies the text of the message. The *message-text* is limited to 512 bytes, after variable substitution. Messages larger than 512 bytes after variable substitution will be truncated when displayed.

Description

The MSG tag defines messages within a message member.

Conditions

You must code the MSG tag within the MSGMBR tag.

Nested Tags

You can code the following tag within the MSG tag:

Tag	Dialog Element	Usage	Page
VARSUB (variable substitution)	Message Member	Multiple	15-190

Example

The following message markup will display in a message pop-up as shown in Figure 15-32.

```
<!doctype dm system>

<msgmbr name=msgA99>
   <msg suffix=0 msgtype=action>You must enter a customer name.
</msgmbr>
```

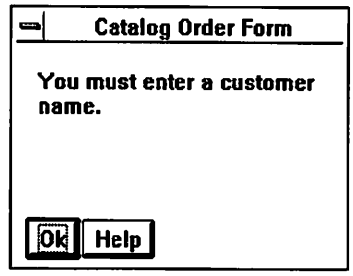
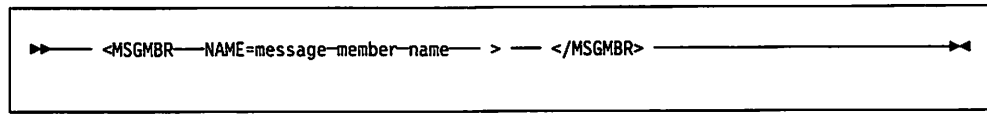


Figure 15-32. Message

MSGMBR (Message Member)

The MSGMBR tag defines a message member.



NAME = message-member-name

The NAME attribute specifies the name of the message member, which also serves as the prefix for all identifiers of messages within the member. The name consists of three upper- or lowercase alphabetic characters, one alphabetic character (A – F only), and two numeric characters (0 – 9). For example, MSGA12 would be a valid name, but MSGXFG, MGA12, and MSGA123 would not.

Lowercase characters are translated to their uppercase equivalents.

Description

The MSGMBR tag defines a message member. The name of a message member is an explicit part of the identifier for a message.

The message member can contain up to 10 messages.

Conditions

None.

Nested Tags

You can code the following tag within the MSGMBR tag:

Tag	Dialog Element	Usage	Page
MSG (message)	Message Member	Multiple	15-114

Example

The following message member markup contains ten messages. Figure 15-33 on page 15-118 shows how the message pop-up for message *maia002* would appear over the panel for which it is displayed.

```
<!doctype dm system>

<varclass name=a type='char 10'>
<varclass name=b type='mixed 15'>
<varclass name=c type='char 8'>

<varlist>
  <vardcl name=a varclass=a>
  <vardcl name=b varclass=b>
  <vardcl name=c varclass=c>
</varlist>
<msgmbr name=maia00>
  <msg suffix=0 msgtype=action>You cannot type a number in the Name field.
  <msg suffix=1 msgtype=action>Please include your first name in the
                               Name field.
  <msg suffix=2 msgtype=action>You must enter a customer name.
  <msg suffix=3 msgtype=action>Unrecognized character in Address field.
                               Please correct.
  <msg suffix=4 msgtype=action>You cannot type a number in the City field.
  <msg suffix=5 msgtype=action>Unrecognized character in City field.
                               Please correct.
  <msg suffix=6 msgtype=action>You cannot type a number in the State field.
  <msg suffix=7 msgtype=action>You must type two letters in the State Field.
  <msg suffix=8 msgtype=action>The Zip Code exceeds the maximum length.
  <msg suffix=9 msgtype=action>You cannot type an alphabetic character
                               in the Zip field.
</msgmbr>

<panel name=msgmbr1 depth=20 width=50>Catalog Order Form

  <area>
    <dtafld datavar=a pmtwidth=10 required=yes entwidth=25
    msg=maia002>Name
    <divider>
    <dtafld datavar=b pmtwidth=15 entwidth=25>Address
    <divider>
    <dtafld datavar=c pmtwidth=10 entwidth=25 >City
    </area>
</panel>
```

The message identifiers used to reference each of the messages are:

```
maia000
maia001
maia002
maia003
maia004
maia005
maia006
maia007
maia008
maia009
```

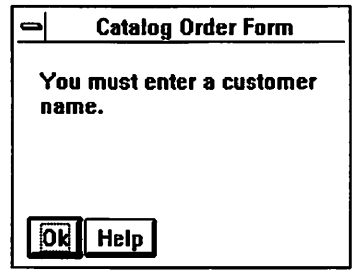
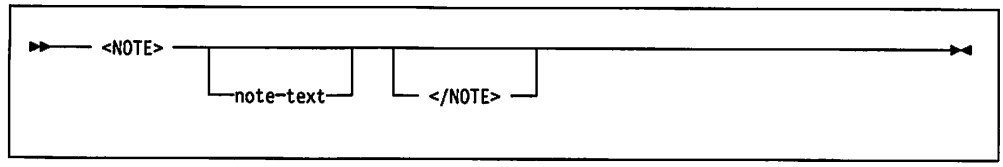


Figure 15-33. Message Member

NOTE (Note)

The NOTE tag defines a single-paragraph note.



note-text
This specifies the text of a note.

Description

The NOTE tag defines a single-paragraph note. The start of another tag ends a note.

The text of the note formats as an implied paragraph, at the left margin. The word "Note: ", or its translated equivalent begins the paragraph. It aligns with the text of a list item when you use it within a list.

Conditions

You can use the NOTE tag only within an information region.

You cannot nest notes.

Nested Tags

You can code the following tag within the NOTE tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Example

In the following panel markup, a NOTE definition is used within a simple list. Figure 15-34 shows the formatted result.

```
<panel name=note1 width=55 depth=20>Departments on the Fifth Floor
  <area>
  <info>
    <p>The following departments are located on the fifth floor:
    <sl>
      <li>Electronics
      <note>
      Don't miss our fabulous prices on electronics.
      No one sells televisions, VCRs and stereos for less!
      <li>Records and tapes
    </sl>
  </info>
  </area>
</panel>
```

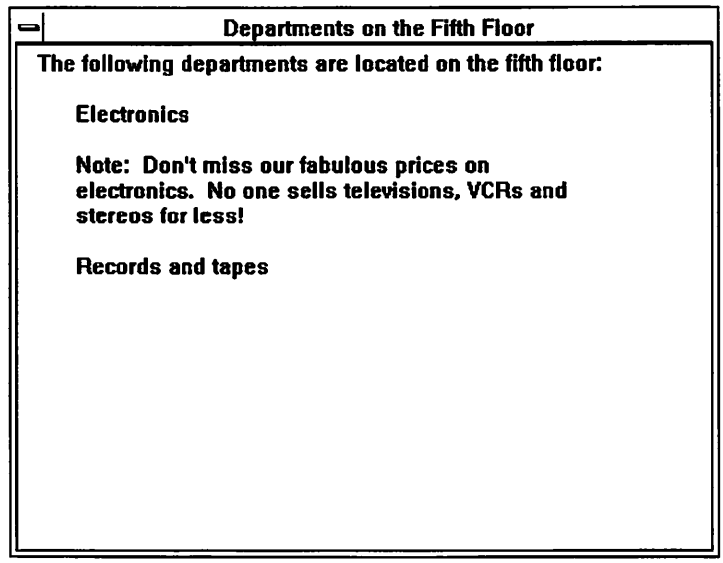
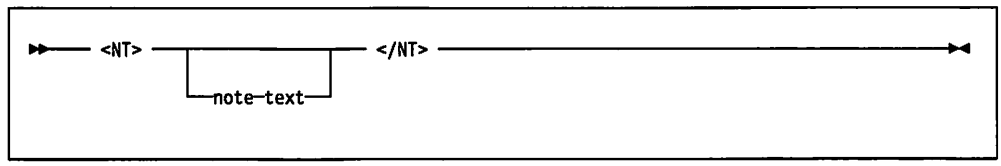


Figure 15-34. Note

NT (Note)

The NT tag defines a single- or multiple-paragraph note.



note-text
The text of the note. You can use the P tag to code multiple paragraphs.

Description

The NT tag defines a single- or multiple-paragraph note.

The text of the note formats as an indented block. The word "Note: ", or its translated equivalent begins the paragraph. It aligns with the text of a list item when you use it within a list.

Conditions

You can code the NT tag only within an information region.

Basic text tags such as paragraph and list tags can follow, but you cannot nest NT and NOTE tags.

Nested Tags

You can code the following tags within the NT tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel	Multiple	15-70
LINES (lines)	Application Panel	Multiple	15-95
OL (ordered list)	Application Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel	Multiple	15-165
UL (unordered list)	Application Panel	Multiple	15-176
XMP (example)	Application Panel	Multiple	15-202

Example

In the following application panel markup, the NT definition is nested within the second list item of a simple list. Figure 15-35 on page 15-122 shows the formatted result.

```
<panel name=nt1 width=55 depth=20>Breeds of Dogs
  <area>
  <info>
    <p>We have the following dogs available:
    <sl>
      <li>Shetland Sheepdogs
      <li>Great Danes
      <nt>Great Danes are very friendly and loyal but we do
      not recommend them for apartment dwellers.
      <p>A Chihuahua is more suitable for apartment dwellers.
      </nt>
      <li>German Shepherds
      <li>Chihuahuas
    </sl>
  </info>
  </area>
</panel>
```

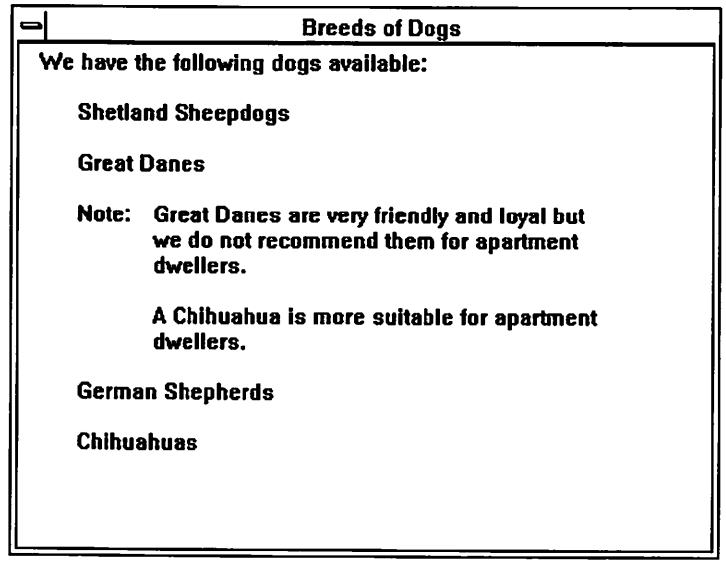
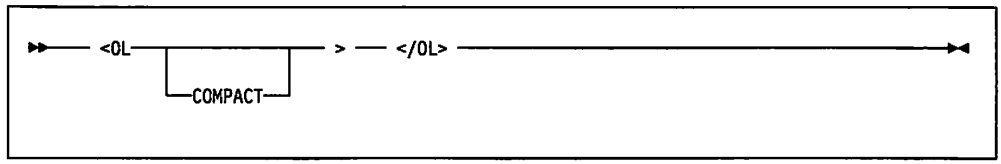


Figure 15-35. Note

OL (Ordered List)

The OL tag defines an ordered list of items.



COMPACT
The COMPACT attribute causes the list to be formatted without a blank line between the items.

Description

The OL tag defines an ordered list of items. You use ordered lists to indicate a set of sequential items or steps.

Ordered lists are formatted as indented lists, with the sequential numbers or letters at the left margin. Nested lists (lists embedded within other lists) indent four spaces to the right of the left margin of the list that contains them. The compiler adds a blank line before the first item in the list.

There are four levels of item identifiers that can appear in an ordered list:

1. Level one
 a. Level two
 1) Level three
 a) Level four

Sequential numbers or letters, depending on the nesting level of the ordered list, are placed ahead of each item in the list. Nesting beyond four levels repeats the sequence starting at level one.

Use the LI (list item) tag to denote each list item. See page "LI (List Item)" on page 15-92 for additional information on how to define ordered lists.

Conditions

You can code the OL tag only within an information region.

Nested Tags

You can code the following tags within the OL tag:

Tag	Dialog Element	Usage	Page
LI (list item)	Application Panel Help Panel	Multiple	15-92
LP (list part)	Application Panel Help Panel	Multiple	15-99

Example

In the following markup, an ordered list is being used to show a logical series of steps needed to complete an action. Figure 15-36 shows the formatted result.

```
<help name=ol1>Help for using the Cash Card
  <area>
  <info>
    <p>To use the Cash Card machine, do the following:
    <ol>
      <li>Insert Cash Card into slot
      <ol>
        <li>Turn the card so the magnetic strip faces up.
        <li>Slide in the card until you feel a slight resistance.
      </ol>
      <li>Enter your password.
      <li>Choose a transaction.
    </ol>
  </info>
  </area>
</help>
```

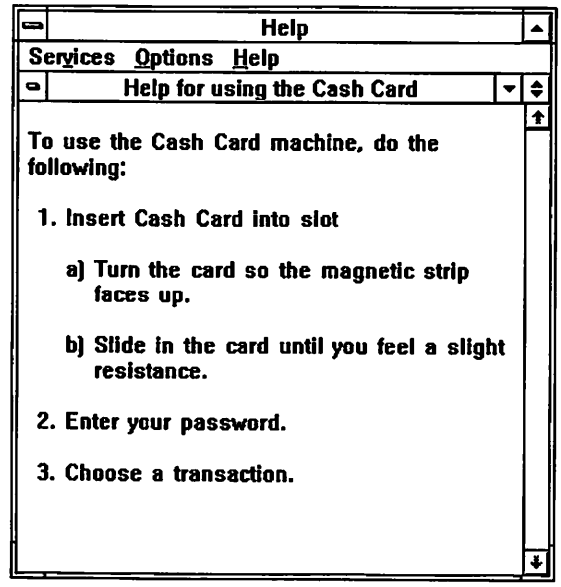
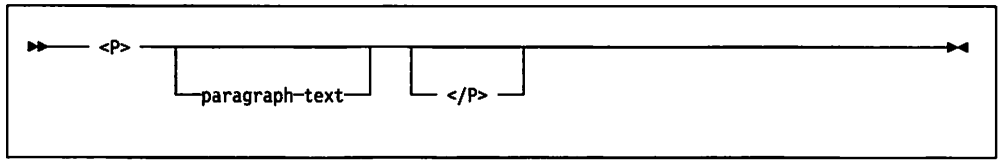


Figure 15-36. Ordered list

P (Paragraph)

The P tag defines a paragraph.



paragraph-text
Paragraph-text specifies the text of the panel.

Description

The P tag defines a paragraph.

Each paragraph formats as an unindented block of text. The compiler adds a blank line before the paragraph.

Conditions

You can use the P tag only within an information region.

Nested Tags

You can code the following tags within the P tag:

Tag	Dialog Element	Usage	Page
CAUTION (caution)	Application Panel Help Panel	Multiple	15-20
RP (reference phrase)	Help Panel	Multiple	15-152
WARNING (warning)	Application Panel Help Panel	Multiple	15-192

Example

In the following markup, there are two paragraphs of text defined. Figure 15-37 on page 15-126 shows the formatted result.

```
<help name=p1>Help for Date Field
  <area>
  <info>
    <p>The application displays the current date
    when the panel is displayed.
    <p>To change the date,
    position the cursor on the Date field
    and type the new date.
    Press Enter to save the new date.
  </info>
  </area>
</help>
```

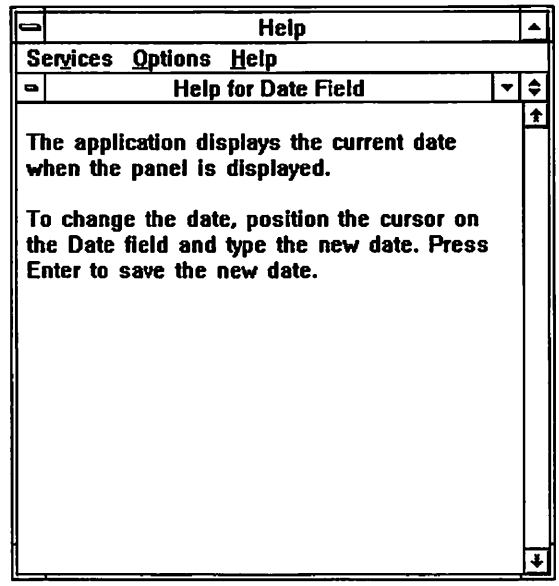


Figure 15-37. Paragraphs

The following markup illustrates the nesting characteristics of the paragraph tag. The paragraph tag nested within the unordered list item assumes the nesting level of the list item. Figure 15-38 on page 15-128 shows the formatted result.

```
<!doctype dm system>

<varclass name=vc1 type = 'char 50'>
<varclass name=vc2 type = 'char 12'>
<varclass name=vc3 type = 'char 2'>
<varclass name=vc4 type = 'char 3'>

<varlist>
  <vardcl name=curdir varclass=vc1>
  <vardcl name=fname  varclass=vc2>
  <vardcl name=copies varclass=vc3>
  <vardcl name=bin varclass=vc4>
</varlist>

<panel name=p2 depth=22 width=75>Print
  <area>
  <region dir=horiz>
  <divider gutter=12
    <info width=54>
      <p>The following items are needed to print a file:
      <ul>
        <li>The directory name. This is shown in the Directory field.
        <li>File name including the file extension.
        <li>Number of copies.
        <p>If you do not specify number of copies, the
        default is 1.
        <li>Bin number where copies should be sent.
        The bin number is alphanumeric.
      </ul>
    </info>
  </region>
  <divider gutter=2
    <dtafld datavar=curdir usage=out>Directory:
    <dtafld datavar=fname>File name
    <dtafld datavar=copies>Number of copies
      <dtafldd>(1 - 99)
    <dtafld datavar=bin>Bin number
  </area>
</panel>
```

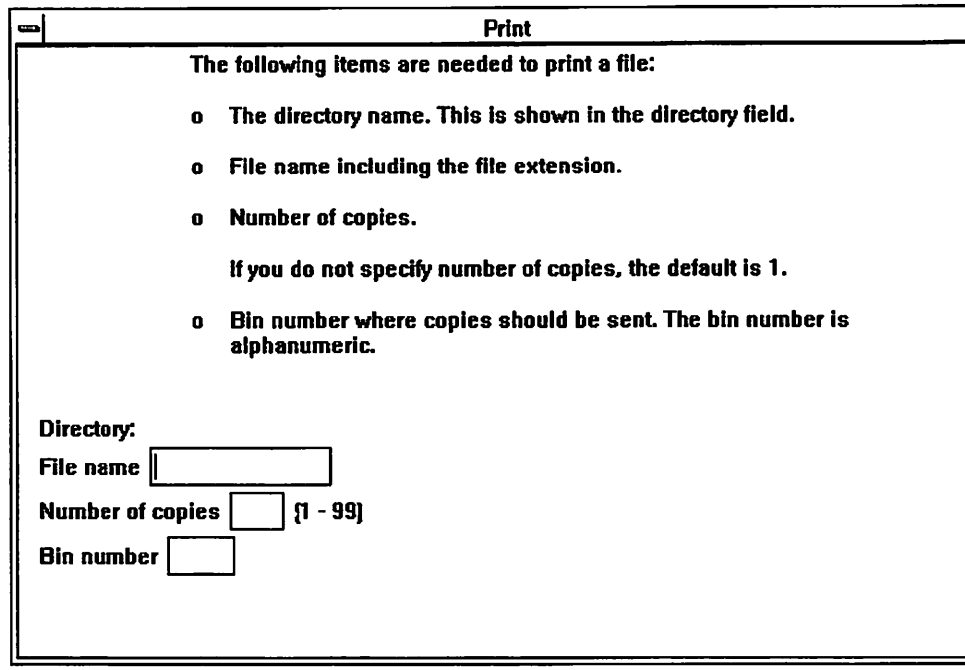
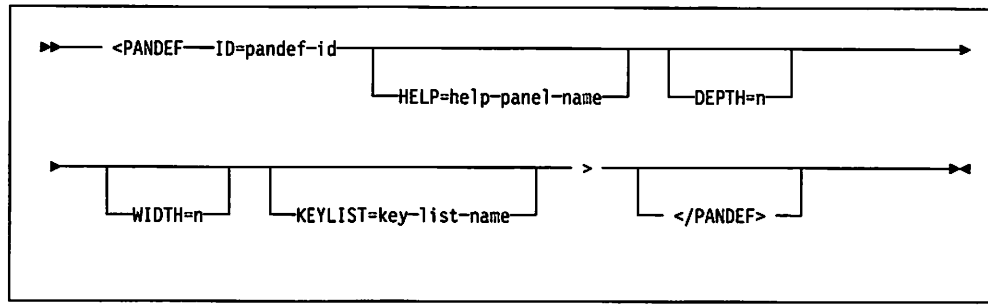


Figure 15-38. Paragraphs

PANDEF (Panel Default)

The PANDEF tag defines defaults for the application panels that reference it.



ID = pandef-id
The ID attribute specifies the identifier of the panel default definition that is referenced from the PANEL tag.

The ID must have the following characteristics:

- 1 – 8 characters in length.
- All characters must be A – Z, a – z, or 0 – 9.

Lowercase characters are mapped to their uppercase equivalents.

HELP = help-panel-name
This specifies the name of a help panel defined using the HELP tag. It identifies the default help text that pertains to panels that reference this PANDEF which do not specify a help attribute. The Dialog Manager displays this panel when the user selects help and the cursor is not on a panel element that has its own help, or when extended help is requested. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents. See "HELP (Help Panel)" on page 15-75 for more information.

DEPTH = n
This specifies the initial depth of the window used to display the panel.

A panel referencing this PANDEF uses this depth unless a different value is specified on the DEPTH attribute of the PANEL tag. The minimum depth value is 1 and the maximum is 512.

See "PANEL (Panel)" on page 15-132 for more information.

WIDTH = n
Specifies the initial width of the window used to display a panel. A panel referencing this PANDEF uses this width unless a different value is specified on the WIDTH attribute of the PANEL tag. The minimum width value is 16 and the maximum is 512.

See "PANEL (Panel)" on page 15-132 for more information.

KEYLIST = key-list-name
This specifies the name of the key mapping list associated with the panel default definition.

Description

The PANDEF tag defines defaults for the application panels that reference it. This information includes:

- The name of the default help panel
- The default dimensions of the window
- The default key mapping list.

Any values you specify on individual PANEL tags take precedence over these values.

Conditions

You cannot code the PANDEF tag within any other tag. It must be coded or embedded before any referencing PANEL tags.

Nested Tags

None.

Example

The following markup shows the same example used in the PANEL tag, but adds the use of the PANDEF tag. By using the PANDEF tag to create the defaults (in this case *depth, width, and keylist*), the PANEL definition can simply reference the PANDEF by *id=maindef*, and add any other attributes. This panel definition specifies its own extended help panel name. The formatted result is shown in Figure 15-39 on page 15-131.

```
<!doctype dm system>
<pandef id=maindef depth=10 width=50 keylist=tappkey>

<panel name=pandef1 pandef=maindef help=mainh>Travel Agency
  <ab>
    <abc><m>File
    <abc><m>View
    <abc><m>Options
    <abc><m>Help
  </ab>
  <topinst>This program allows you to make travel
           arrangements to any part of the world.
  <area>
  </area>
</panel>
```

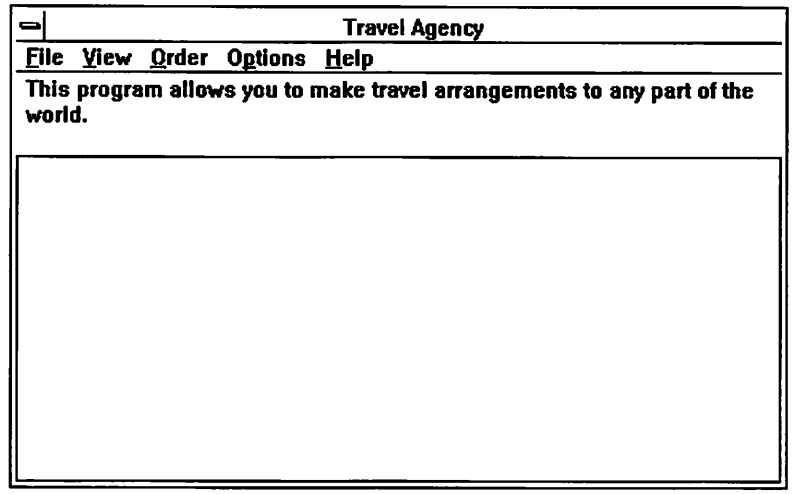
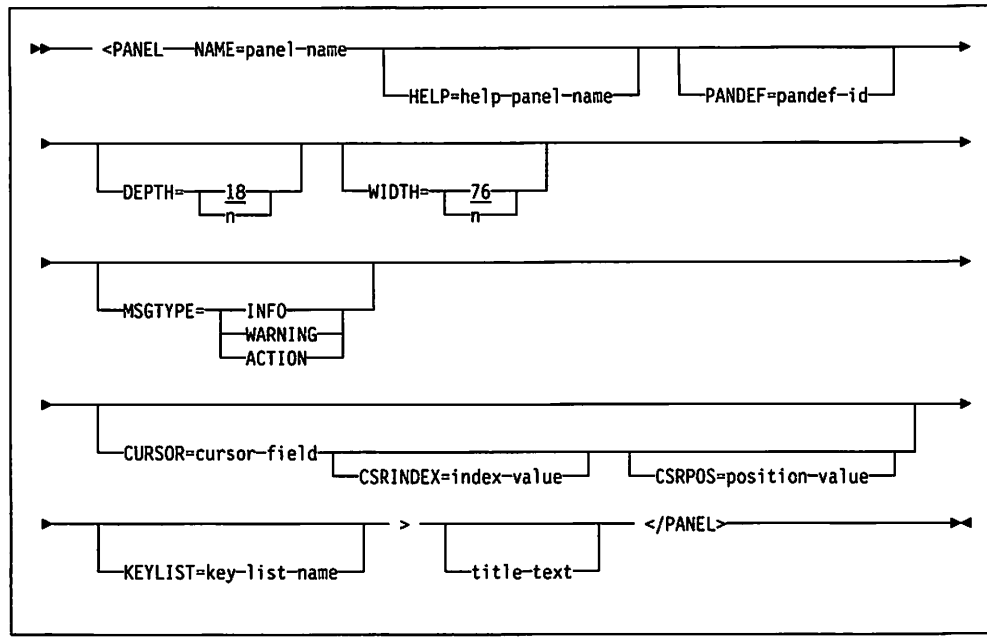


Figure 15-39. Panel default

PANEL (Panel)

The PANEL tag defines an application panel.



NAME = panel-name

This specifies the name of the panel. The name is used as the panel identifier, which the user can display. When displayed, it appears to the left of the window title text, separated from it by a hyphen (—) . The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

This attribute specifies the name of a help panel defined using the HELP tag. It identifies the help text that is associated with this panel definition. The Dialog Manager displays the help panel when the user selects help and the cursor is not on a panel element that has its own help specified. The help panel also appears as the result of a user request for extended help. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents. See "HELP (Help Panel)" on page 15-75 for additional details.

PANDEF = pandef-id

This attribute specifies the set of panel defaults to be used. The *pandef-id* is the identifier specified on the PANDEF tag. You can override any of the defaults from this PANDEF tag by specifying that attribute on the PANEL tag.

DEPTH = 18 | n

The DEPTH attribute specifies the initial depth (in character units) of the panel you are defining. The default depth is 18.

If this panel is the first to be displayed in the window, the DEPTH attribute is used to define the window depth. If it is a primary window, the user can size the window after it is displayed.

If the panel is displayed in an existing window, this attribute is ignored and the depth of the current window is used.

The value specified for the depth does not include the title, the action bar, the function key area, or the borders. The depths of these items are added to the depth value specified. If the depth specified is too big to fit in the currently displayed window, the bottom of the panel will not be visible.

The range is 1 to 512.

WIDTH = 76 | n

The WIDTH is the initial width of the window (in character units) for the panel you are defining. The default width is 76.

If this panel is the first to be displayed in the window, the WIDTH attribute is used to define the window width. If it is a primary window, the user can resize the window when it is displayed.

If the panel is displayed in an existing window, this attribute is ignored and the width of the current window is used.

The value specified for the width is the width of the entire window, not including the borders. If the width specified is too big to fit in the window, then the right-most side of the panel will not be visible.

The range is 16 to 512.

MSGTYPE = INFO | WARNING | ACTION

INFO indicates an information message, WARNING indicates a warning message, and ACTION indicates an action message.

The Dialog Manager sounds an audible alarm for WARNING and ACTION messages.

Note: You can use the word ERROR as a substitute for ACTION in the MSGTYPE attribute.

This attribute is useful when you are defining your own message panels. When you use the PANEL tag to define your own message panels, use the MSGTYPE attribute to tell Dialog Manager which type of message will be displayed in the panel.

CURSOR = cursor-field
This attribute, together with CSRINDEX and CSRPOS, controls the initial placement of the cursor when the Dialog Manager displays the panel. You may specify, for this attribute, the value of the NAME attribute from one of the following tags:

- CHOICE
- DTAFLD
- LSTCOL
- LSTFLD
- SELFLD
- SELLST
- UC

The cursor can also be placed on the command area, when it exists on a panel. Use the DM-reserved name *cmdarea* as the value for the *cursor-field* to place the cursor on the command area.

CSRINDEX = index-value
This attribute, together with CURSOR and CSRPOS, controls the placement of the cursor when the Dialog Manager displays this panel. CSRINDEX specifies the row into the array where the Dialog Manager places the cursor when it displays the panel.

The first index of an array field is denoted by 1. The maximum value that you can specify is the current dimension of the array.

This attribute may be specified only when the CURSOR attribute references a LSTCOL tag.

This attribute only applies to list fields.

CSRPOS = position-value
This attribute, together with CURSOR and CSRINDEX (on a list column only), controls the placement of the cursor when the Dialog Manager displays this panel. CSRPOS specifies the number of byte positions into the entry field that the Dialog Manager places the cursor when it displays the panel.

The first position of a field is denoted by 1. The maximum position that you can specify is the length of the underlying data.

This attribute may be specified only when the CURSOR attribute references a DTAFLD or a LSTCOL tag, or the command area.

If the value specified for this attribute is not valid, the default (1) is used.

KEYLIST = key-list-name
This attribute specifies the name of the key mapping list associated with the panel. You specify the default on the PANDEF tag. If you do not specify a default on the PANDEF tag, the default is the key mapping list supplied with Dialog Manager.

title-text
This specifies the title when the panel is displayed. You can override this value at run time by setting the system variable, ZWINTTL. ZWINTTL can only override *title-text* for the primary window.

If the *title-text* exceeds 255 bytes, the compiler truncates the text and issues a warning. The title is further truncated if there is not enough room to display it in the title bar.

Description

The PANEL tag defines an application panel.

The panel definition consists of:

- The panel name
- The name of the help panel that pertains to the panel as a whole (extended help)
- The dimensions
- A means of controlling cursor placement on the panel
- The window title text.

You can use the related attributes on the DISPLAY service to override the cursor positioning attributes.

If the panel contents are larger than the WIDTH or DEPTH on the PANEL tag, you must code an AREA tag to prevent the panel contents from being truncated.

If you specify the PANDEF attribute, the values specified in the PANDEF definition override the default values for unspecified PANEL attributes.

Conditions

You cannot code a PANEL tag within any other tag.

A panel must contain a panel body; otherwise, an error message will be issued. The panel must contain at least one of the following tags:

- BOTINST
- DTAFLD
- INFO
- LSTFLD
- SELLST
- SELFLD
- TOPINST
- UC

Nested Tags

You can code the following tags within the PANEL tag:

Tag	Dialog Element	Usage	Page
AB (action bar)	Application Panel	Single	15-3
AREA (area)	Application Panel Help Panel	Single	15-11
BOTINST (bottom instruction)	Application Panel	Multiple	15-18
CMDAREA (command area)	Application Panel	Single	15-40
DIVIDER (area divider)	Application Panel	Multiple	15-48
DTACOL (data column)	Application Panel	Multiple	15-56
DTAFLD (data field)	Application Panel	Multiple	15-60
INFO (information)	Application Panel Help Panel	Multiple	15-82
LSTFLD (list field)	Application Panel	Multiple	15-106
REGION (region)	Application Panel	Multiple	15-149
SELLST (selection list)	Application Panel	Multiple	15-161
SELFLD (selection field)	Application Panel	Multiple	15-156

Tag	Dialog Element	Usage	Page
TOPINST (top instruction)	Application Panel	Multiple	15-171
UC (user control)	Application Panel	Multiple	15-173

Example

The following markup of an application panel contains only an action bar and a top instruction. The keys that will be active when this panel is displayed are defined in the key mapping list, *tappkey*. The initial width and depth are specified, and the extended help panel, *mainh*, is also named. The title text is "Travel Agency," which is the name of the application. Figure 15-40 shows the formatted result.

```
<!doctype dm system>
<keyl name=tappkey>
</keyl>
<panel name=panel1 help=mainh depth=10 width=50
       keylist=tappkey>Travel Agency
  <ab>
    <abc><m>File
    <abc><m>View
    <abc><m>Options
    <abc><m>Help
  </ab>
  <topinst>This program allows you to make travel
           arrangements to any part of the world.
  <area>
  </area>
</panel>
```

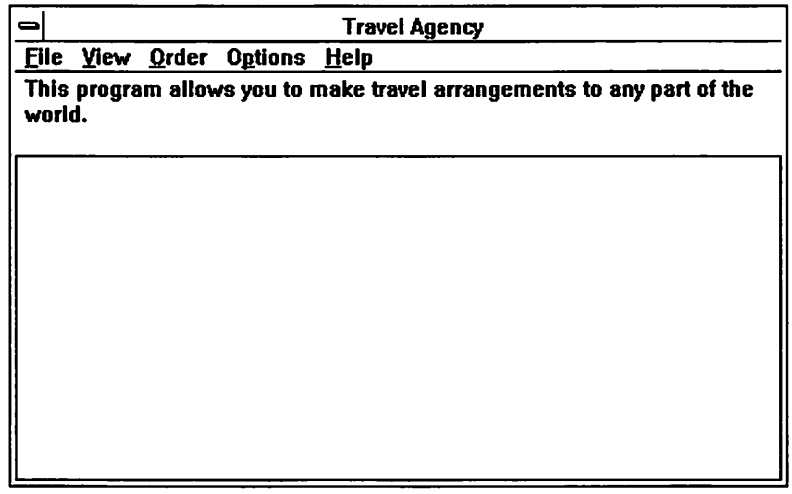
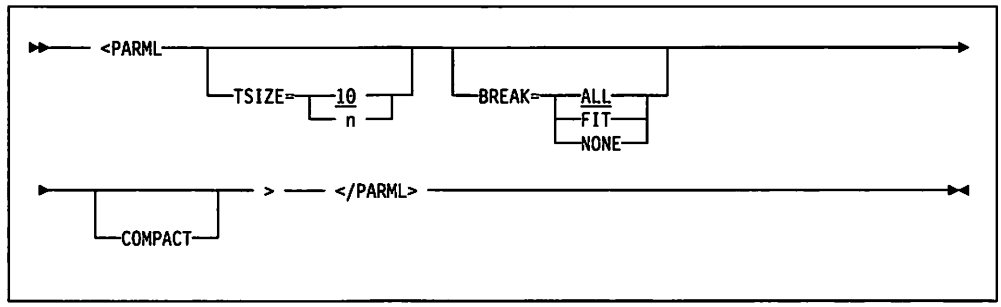


Figure 15-40. Application Panel

PARML (Parameter List)

The PARML tag starts a list of parameter terms and descriptions.



TSIZE = 10 | n
The TSIZE attribute specifies the space allocated for the parameter term. The minimum space allotted is 1 and the maximum is 128 character units. The default is 10 character units.

BREAK = ALL | FIT | NONE
The BREAK attribute controls the formatting of the parameter terms and descriptions. If BREAK = ALL, every description is on the line below the term. If BREAK = FIT, and the term is longer than the TSIZE value, the description will appear on the line below the term. If BREAK = NONE, and the term is longer than the TSIZE value, the term will appear on the same line as the description, spilling into the description area. The description will start one space after the term.

COMPACT
The COMPACT attribute formats the list without a blank line between the items.

Description

The PARML tag starts a list of parameter terms and descriptions.

Parameter lists are similar to definition lists. They involve three tags: PARML (parameter list) and a matching end tag, PT (parameter term), and PD (parameter description). As in definition lists, the term tag identifies a term, and the definition tag identifies the description associated with the term. The PD tag must immediately follow the PT tag that it is associated with.

Conditions

You can code the PARML tag only within an information region.

Nested Tags

You can code the following tags within the PARML tag.

Tag	Dialog Element	Usage	Page
PD (parameter description)	Application Panel Help Panel	Multiple	15-140
PT (parameter term)	Application Panel Help Panel	Multiple	15-147

Example

The following markup contains a parameter list that provides a description of inventory part numbers. A second parameter list is nested within the third parameter description. Figure 15-41 on page 15-139 shows the formatted result.

```
<panel name=parml1 width=50 depth=25>Coding for Part Numbers
  <area>
  <info>
    <p>Part numbers are composed of a five-digit number.
    The part number consists of three components.
    <p>For example,
    <xmp>
        12345AA
    </xmp>
    <parml tsize=6>
      <pt>123
      <pd>The first three digits represent the
      lot number of the part.
      <pt>45
      <pd>The fourth and fifth digits represent the
      date code of the part.
      <pt>AA
      <pd>The two-character suffix represents the department
      the part originated from.
      The acceptable suffixes are:
      <parml break=none compact>
        <pt>TO
        <pd>Tools
        <pt>EL
        <pd>Electrical
        <pt>ME
        <pd>Mechanical
      </parml>
    </parml>
  </info>
  </area>
</panel>
```

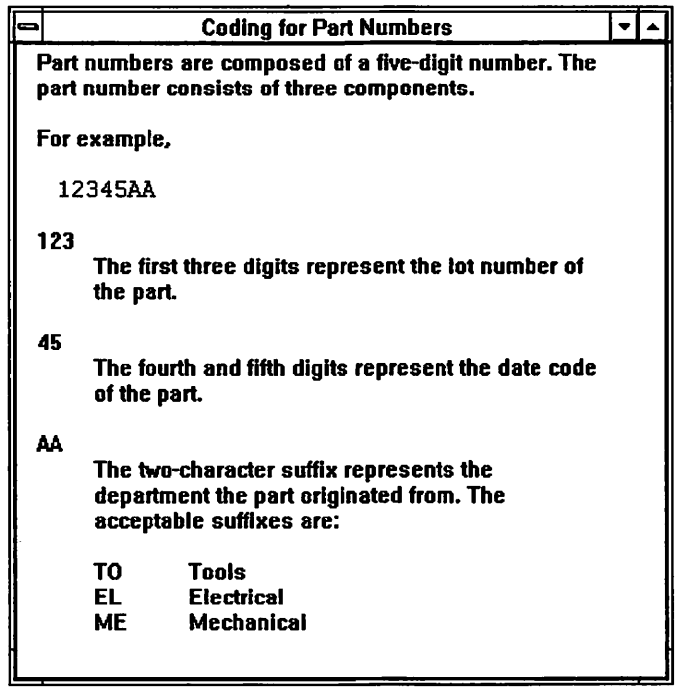
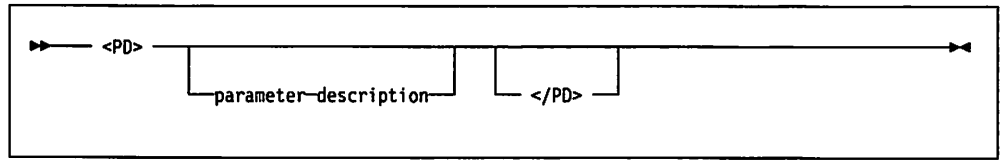


Figure 15-41. Parameter List

PD (Parameter Description)

The PD tag defines the description for the term being defined in a parameter list.



parameter-description
The *parameter-description* contains the text of a parameter description.

Description

The PD tag defines the description for the term being defined in a parameter list.

The column in which the description starts is determined by the TSIZE attribute specified on the PARML tag. See "PARML (Parameter List)" on page 15-137 for more information.

Conditions

You can use the PD tag only within an information region.

The PD tag is valid only within a parameter list. For more information, see "PARML (Parameter List)" on page 15-137.

Each PD tag must be paired with a PT tag. You can specify only one PD tag for each PT tag within a parameter list. The PD tag must immediately follow the PT tag that it is associated with.

Nested Tags

You can code the following tags within the PD tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165
UL (unordered list)	Application Panel Help Panel	Multiple	15-176

Tag	Dialog Element	Usage	Page
XMP (example)	Application Panel Help Panel	Multiple	15-202

Example

The following markup shows three parameter descriptions within a parameter list, each of them matched with an associated parameter term. Figure 15-42 shows the formatted result.

```
<help name=pdl>Help for Ordering Parts
  <area>
  <info>
    <p>Use one of the following codes when ordering
    a part number from inventory:
    <parml tsize=6>
      <pt>ST
      <pd>Indicates that the part order is for
      stock replenishment.
      <pt>CU
      <pd>Indicates that the part order is
      for immediate customer shipment.
      <pt>EL
      <pd>Indicates that the part order is
      for shipment to an external location.
    </parml>
  </info>
  </area>
</help>
```

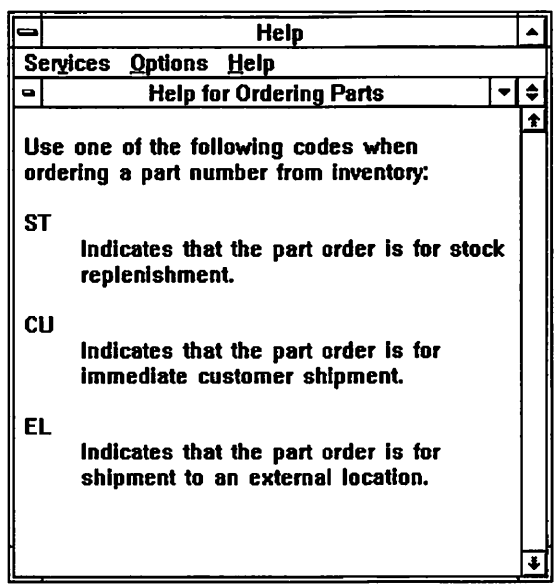
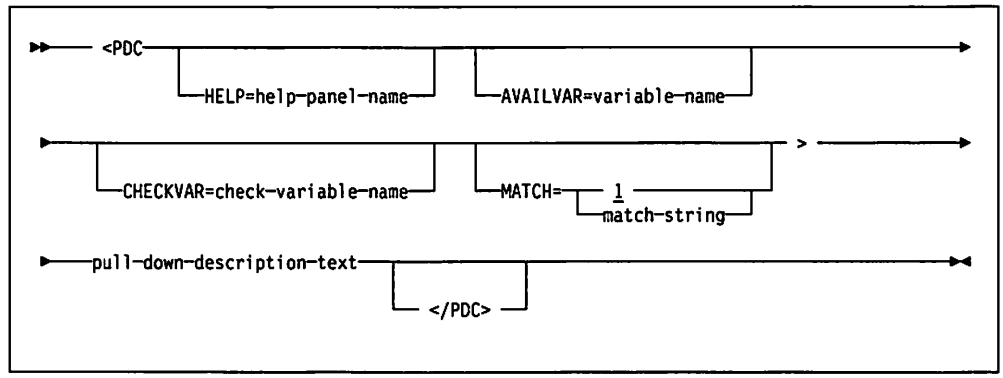


Figure 15-42. Parameter Descriptions

PDC (Pull-Down Choice)

The PDC tag defines information about a choice included on an action bar pull-down.



HELP = help-panel-name
The name of the help panel the Dialog Manager displays when the user selects help for this pull-down choice.

If the user selects help on a pull-down choice, the help defined for the choice is displayed. If no help was defined for the pull-down choice, the help for the action bar choice is displayed. If there is no help defined for the action bar choice, the extended help panel is displayed. If an extended help panel was not defined, a message is displayed stating that no help is defined. The name must have the following characteristics:

- 1–8 characters in length.
- The first (or only) character must be A–Z or a–z.
- Remaining characters, if any, must be A–Z, a–z, or 0–9.

Lowercase characters are translated to their uppercase equivalents.

AVAILVAR = variable-name
The AVAILVAR attribute specifies the availability of the pull-down choice. If the value of the specified *variable-name* is true (1) during display, the pull-down choice is available. A value of false (0) indicates that the pull-down choice is unavailable.

The attribute value must be coded without using % notation. If you do not specify this attribute, the pull-down choice is always available. You must ensure that the value of the named variable correctly relates to the availability of this action. An unavailable choice is indicated by displaying the choice de-emphasized to show that it is unavailable.

Often, actions for pull-down choices are implemented by issuing a command when the pull-down choice is selected. The command to issue is named by coding an ACTION tag under the PDC tag. Like pull-down choices, commands can also be made unavailable (through the AVAILVAR attribute on the CMD tag). If you specify an AVAILVAR attribute for both a command and a pull-down choice that references the command, you should be sure to keep the two synchronized.

CHECKVAR = check-variable-name
This is a variable whose value indicates whether the pull-down choice is selected when the pull-down menu is displayed. If the value of the variable is equivalent to the *match-string* you specify with the MATCH attribute, the

pull-down choice has a check mark in front of it. Otherwise, it does not. The CHECKVAR attribute value must be specified without using % notation.

MATCH = 1 | match-string
The MATCH attribute defines the value for the pull-down choice that causes the choice to appear as a check mark. The check value is compared to the match-string value and if they are equal, the pull-down choice appears with a check mark .

pull-down-description-text
This indicates the text of the action bar choice pull-down. The maximum length of the text is 40 bytes.

If the selection of the action bar pull-down choice results in the display of a pop-up, Common User Access requires an ellipsis (...) following the text. You must ensure that the ellipsis is included within the *pull-down-description-text*.

Description

The PDC tag defines information about a choice included on an action bar pull-down. If you do not code any PDC tags within an ABC tag, that action bar choice will appear on the action bar, but no action will be taken when the user selects an action bar item.

Pull-down choices represent immediate actions to the user. Code an ACTION tag, specifying the RUN attribute under a PDC tag, so that an immediate action will occur when the pull-down item is selected by the user. For more information, see "ACTION (Action)" on page 15-7.

Conditions

You can code a PDC tag only within an ABC tag.

To conform to CUA and to enhance portability:

- Specify the CHECKVAR attribute on each PDC tag
- Separate logical groups of choices on a pull-down menu with the use of the PDSEP tag
- All pull-down choices within a group of mutually exclusive choices must name the same variable in the CHECKVAR attribute and have different values for the MATCH values. The user is expected to select a single choice from those provided within the group. When one choice is selected, the other choices in the group will be de-selected.
- All pull-down choices within a group of choices that are not mutually exclusive must name different variables in their CHECKVAR attributes. The user must explicitly de-select choices that are not wanted, since selecting a choice does not automatically de-select other choices in the group.

Nested Tags

You can code the following tags within the PDC tag:

Tag	Dialog Element	Usage	Page
ACTION (action)	Application Panel	Multiple	15-7
M (mnemonic)	Application Panel	Single	15-110

Example

```
<!doctype dm system>

<varclass name=availcls type ='char 1'>
</varclass>

<varlist>
  <vardcl name=nameyet varclass=availcls>
</varlist>

<panel name=pdc1 width=30 depth=10>Catalog Ordering System
  <ab>
    <abc><m>File
      <pdc><m>New
        <action run=New>
      <pdc><m>Open...
        <action run=Open>
      <pdsep>
      <pdc availvar=nameyet><m>Save
        <action run=Save>
      <pdc>Save <m>as...
        <action run=SaveAs>
      <pdsep>
      <pdc>E<m>xit
        <action run=Exit>
    </abc>

    <abc><m>View

    <abc><m>Options

    <abc><m>Help
  </ab>
  <info>
  </info>
</panel>
```

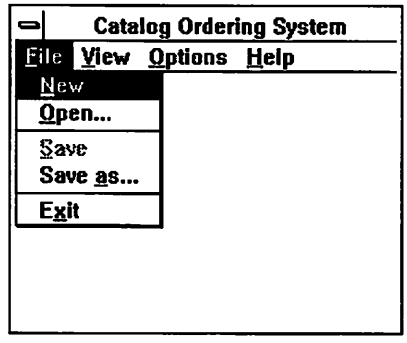
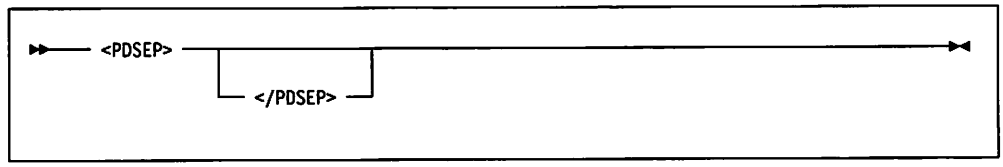


Figure 15-43. Pull-Down Choices

PDSEP (Pull-Down Separator)

The PDSEP tag defines a horizontal divider line on an action bar pull-down.



Description

The PDSEP tag defines a horizontal divider line on an action bar pull-down.

You use the horizontal divider to show the separation between groups of related pull-down choices.

Conditions

You can code a PDSEP tag only within an ABC tag.

Nested Tags

None.

Example

The following markup shows how the PDSEP tag is used to draw a separator in the Help pull-down. Figure 15-44 on page 15-146 shows the formatted result.

```
<!doctype dm system>

<panel name=pdsep1 width=50 depth=10>Catalog Ordering System
  <ab>
    <abc><m>File

    <abc><m>View

    <abc><m>Options

    <abc><m>Help
     <pdc><m>Help for help...
       <action run=helphelp>
     <pdc><m>Extended help...
       <action run=exhelp>
     <pdc><m>Keys help...
       <action run=keys>
     <pdc>Help <m>index...
       <action run=index>
     <pdc><m>Tutorial...
       <action run=tutorial>
     <pdsep>
     <pdc><m>About...
       <action run=logo>
    </abc>
  </ab>
  <info>
  </info>
</panel>
```

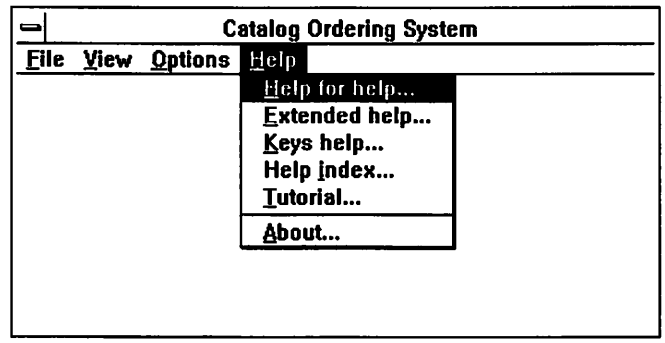
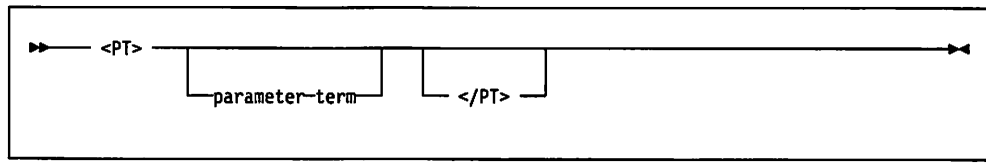


Figure 15-44. Pull-Down Separator

PT (Parameter Term)

The PT tag defines a term in a parameter list.



parameter-term
This attribute specifies the text of the parameter term.

Description

The PT tag defines a term in a parameter list.

You can code multiple PT tags within a PARML (parameter list) tag.

Conditions

You can use the PT tag only within an information region.

The PT tag is valid only within a parameter list. See "PARML (Parameter List)" on page 15-137 for more information.

The description for the term is specified with the PD (parameter description) tag.

Nested Tags

You can code the following tag within the PT tag:

Tag	Dialog Element	Usage	Page
RP (reference phrase)	Help Panel	Multiple	15-152

Example

The following help panel markup shows two parameter terms within a parameter list, each of them matched with an associated parameter description. Figure 15-45 on page 15-148 shows the formatted result.

```
<help name=pt1>Help for the Duplex Function
  <area>
  <info>
    <p>The two options associated with
    the DUPLEX function are:
    <parml break=none>
      <pt>DCopies
      <pd>Which prints one-sided copies that
      are prepared for future duplex copying.
      <pt>DPrint
      <pd>Which prints two-sided copies.
    </parml>
  </info>
  </area>
</help>
```

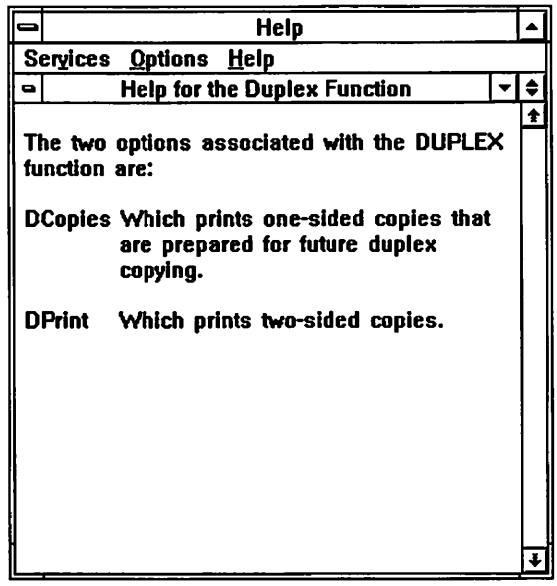
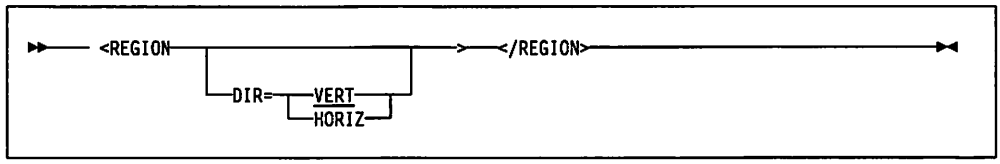


Figure 15-45. Parameter Terms

REGION (Region)

The REGION tag defines the direction in which fields on a panel are arranged.



DIR = VERT | HORIZ

The DIR attribute specifies in which direction fields defined within the region are arranged. If you do not specify the direction, the region is formatted vertically.

If you code DIR = VERT, the region is laid out top to bottom.

If you code DIR = HORIZ, the region is laid out left to right.

Description

The REGION tag defines the direction in which fields on a panel are arranged. The region is sized around the field or fields it contains. The region does not add any additional space. You code these fields using other tags.

Conditions

You can code this tag within an AREA, PANEL or REGION tag for an application panel.

Nested Tags

You can code the following tags within the REGION tag on an application panel:

Tag	Dialog Element	Usage	Page
DIVIDER (area divider)	Application Panel	Multiple	15-48
DTACOL (data column)	Application Panel	Multiple	15-56
DTAFLD (data field)	Application Panel	Multiple	15-60
INFO (information)	Application Panel	Multiple	15-82
LSTFLD (list field)	Application Panel	Multiple	15-106
REGION (region)	Application Panel	Multiple	15-149
SELFLD (selection field)	Application Panel	Multiple	15-156
SELLST (selection list)	Application Panel	Multiple	15-161
UC (user control)	Application Panel	Multiple	15-173

Example

The following markup shows two horizontal regions and a vertical region. Notice that regions can be nested; the region containing the INFO tag is nested inside a horizontal region. Also note that the AREA and SELFLD tags are implicit vertical regions. You do not have to code a vertical region when using those tags. Figure 15-46 on page 15-151 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type = 'char 15'>
<varclass name=statcls type = 'char 2'>
<varclass name=zipcls  type = 'char 5'>
<varclass name=datecls type = idate>

<varlist>
  <vardcl name=name     varclass=sampcls>
  <vardcl name=addr     varclass=sampcls>
  <vardcl name=city     varclass=sampcls>
  <vardcl name=stat     varclass=statcls>
  <vardcl name=zip      varclass=zipcls>
  <vardcl name=graddate varclass=datecls>
  <vardcl name=major    varclass=sampcls>
</varlist>
<panel name=region1 width=65 depth=22>Application Name
  <topinst>Complete the information below.
  <area>
    <dtacol pmtwidth=7>
      <dtafld datavar=name>Name
      <dtafld datavar=addr>Address
    </dtacol>
    <region dir=horiz>
      <dtafld datavar=city pmtwidth=7>City
      <dtafld datavar=stat>State
      <dtafld datavar=zip>Zip code
    </region>
```

```
    <divider type=solid gutter=3>
    <region dir=horiz>
      <selfld>Highest education level
        <choice>Some high school
        <choice>High school graduate
        <choice>Some college
        <choice>College graduate
        <choice>Some post-graduate work
        <choice>Post-graduate degree
      </selfld>
      <divider>
      <region>
        <info width=30>
          <p>Complete if applicable:
        </info>
        <dtacol pmtwidth=10>
          <dtafld datavar=graddate>Date of graduation
          <dtafld datavar=major>Field of study
        </dtacol>
      </region>
    </region>
  </area>
</panel>
```

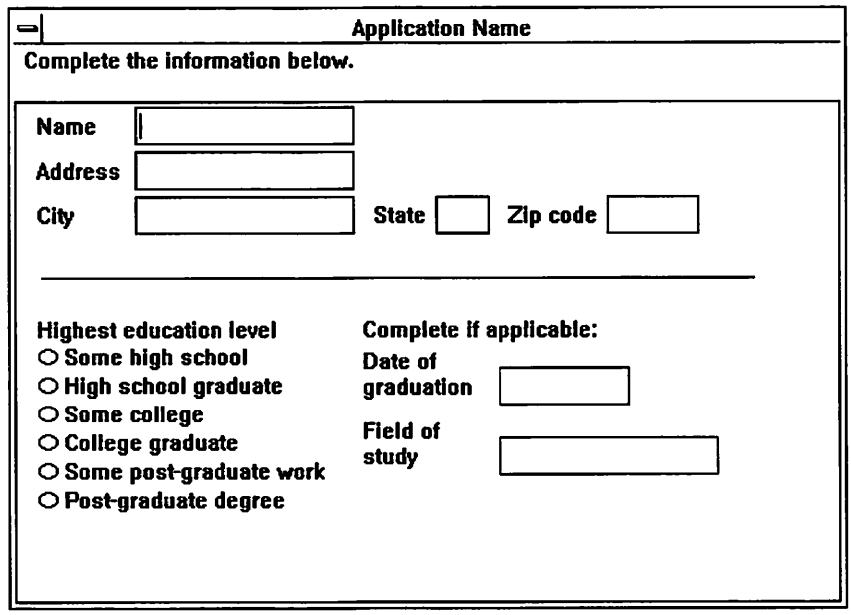
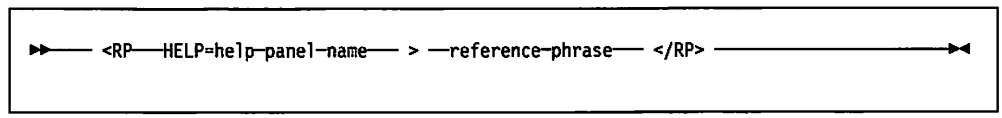


Figure 15-46. Information Region

RP (Reference Phrase)

The RP tag specifies a word or phrase in help text that has additional help information associated with it.



HELP = help-panel-name
This attribute specifies the name of a help panel describing the reference phrase. It identifies the help text that the Dialog Manager displays when the user requests additional information about this reference phrase. See the "HELP (Help Panel)" on page 15-75 for additional information on how to specify the name of a help panel and how to define a help panel.

reference-phrase
This is the text of the phrase. The text is translatable and can be of any length.

Description

The RP tag specifies a word or phrase in help text that has additional help information associated with it.

The reference phrase is emphasized within the text of the help panel to inform the user that additional information is available. The first word of the reference phrase displays in inverse color highlighting to indicate the position of the cursor. The user can select the reference phrase to display the additional information.

Conditions

You cannot use reference phrases in application panels.

The help panel named on the RP tag must be contained within the source file being compiled.

Nested Tags

None.

Example

The following help panel markup contains a reference phrase definition for the phrase, "warranty." When the user selects the reference phrase **warranty**, the help panel specified by the HELP attribute (help=warrh) is displayed. Figure 15-47 on page 15-153 shows the formatted result.

```
<help name=rp1>Help for Appliances
  <info>
    <p>In addition to our free delivery and
    installation program, we also offer an
    exclusive <rp help=warrh>warranty</rp> on all
    of our appliances.
  </info>
</help>

<help name=warrh>Help for Warranty
  <info>
    <p>Warranty covers the replacement of
    any part that breaks or becomes non-functional
    while this product is used by the original owner.
  </info>
</help>
```

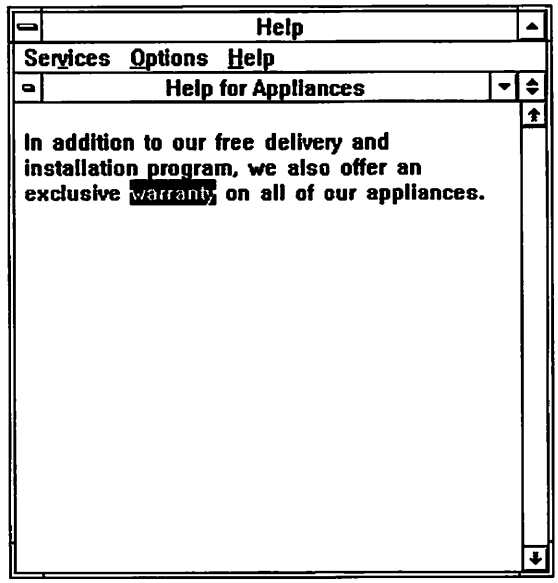
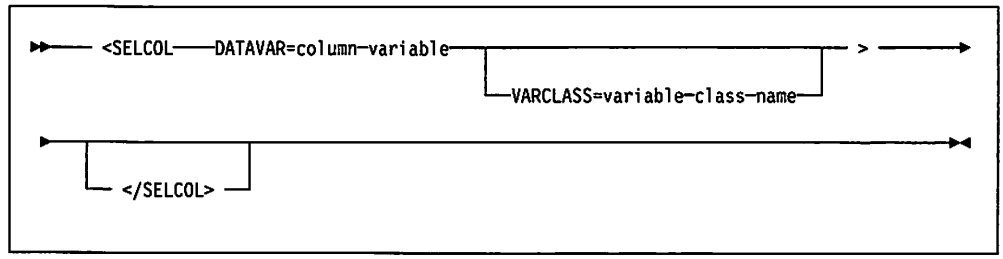


Figure 15-47. Reference Phrase

SELCOL (Selection Column)

The SELCOL tag defines the array of column data where each element of the array corresponds to an item in the selection list.



DATAVAR = column-variable
The DATAVAR attribute specifies the name of the array variable that contains data listed in this selection list column.

VARCLASS = variable-class-name
This attribute specifies the name of the variable class, defined using the VARCLASS tag. This variable class overrides the default variable class referenced by the VARDCL that declared the data variable for this column. The TYPE information of the specified variable class must be the same as the default VARCLASS for DATAVAR.

If the VARCLASS specified does not have TYPE information that matches the type information of the overridden VARCLASS, a warning message is issued .

Description

The SELCOL tag defines the array of column data where each element of the array corresponds to an item in the selection list.

Conditions

The SELCOL tag must be coded only once within the SELLST tag.

Nested Tags

None.

Example

The following markup shows a single-choice selection list. The data shown in the selection list comes from the array data variable, *seafood*, and the **seafood** item selected is returned in the variable, *item*. The number of items in the list is defined by the variable, *numitems*. The selection list size is limited to displaying no more than 10 items. For additional information on the selection variable SELVAR, refer to "SELLST (Selection List)" on page 15-161. Figure 15-48 on page 15-155 shows the formatted result.

```
<!doctype dm system>

<varclass name=foodcls type='char 25'>
<varclass name=numcls  type='numeric 5'>
<varclass name=cnumcls type='char 6'>

<varlist>
  <vardcl name=seafood  varclass=foodcls maxdim=100>
  <vardcl name=item     varclass=numcls>
  <vardcl name=numitems varclass=numcls>
  <vardcl name=custnum  varclass=cnumcls>
</varlist>

<panel name=selcol1>Wholesale Food Distributors
  <topinst>These are the fresh seafood items available this week.
           Select the item that you wish to order, and enter the
           customer number.
  <sellst selvar=item numrows='%numitems' maxrows=10>
    <selcol datavar=seafood>
  </sellst>
  <divider>
  <dtafld datavar=custnum usage=in>Customer Number
</panel>
```

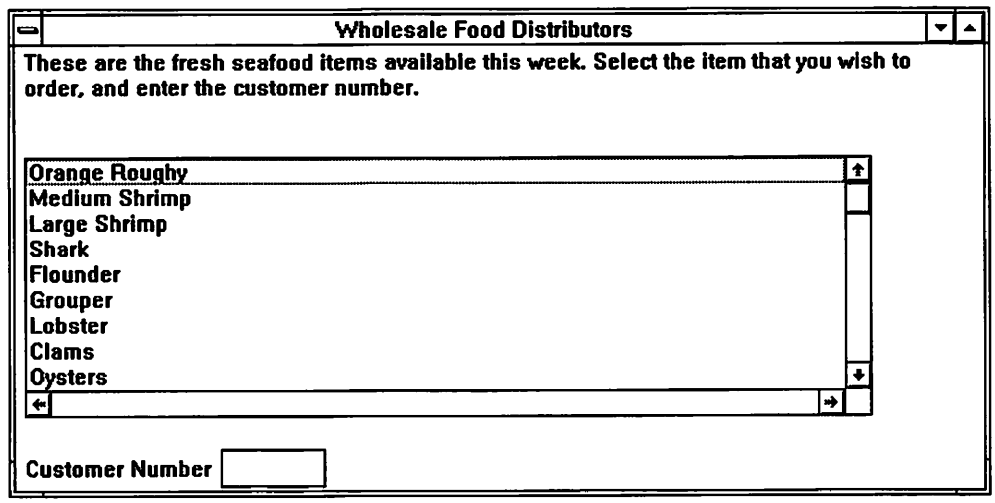
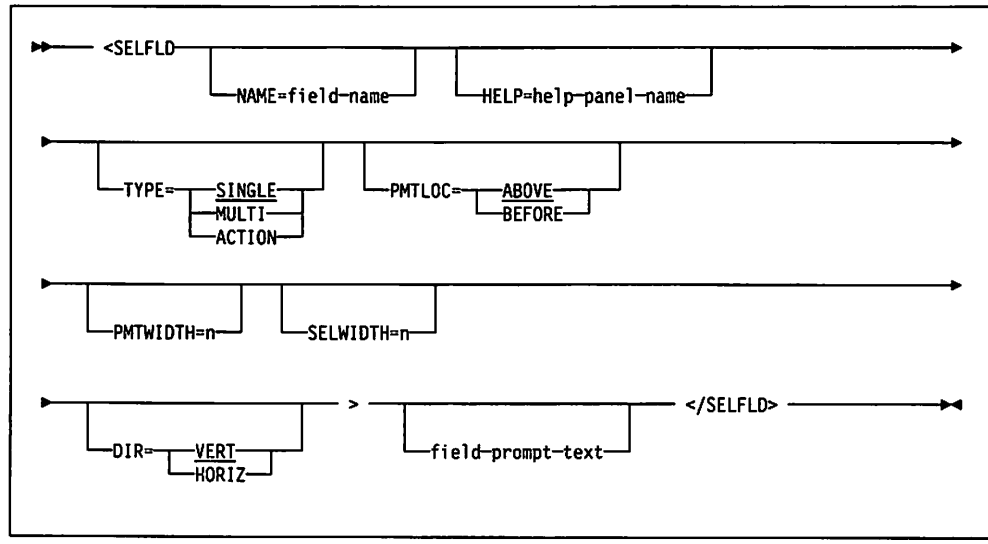


Figure 15-48. Selection Columns

SELFLD (Selection Field)

The SELFLD tag defines a field that includes a list of choices.



NAME = field-name

This attribute specifies the name for the selection field. This name can be used on the DISPLAY call to:

- On the DISPLAY call to position a message pop-up window relative to the selection field.
- On an ADDPOP call to position a pop-up window relative to a selection field.
- On the DISPLAY call to place the cursor in the selection field.
- On the PANEL definition to place the cursor in the selection field.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

The HELP attribute specifies the help panel for a selection field. This help can be superseded by the help specified on the CHOICE tag. If help is not defined for the selection field or on the CHOICE tag, the extended help panel displays. If an extended help panel is not defined, a message displays stating that no help is defined. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

TYPE = SINGLE | MULTI | ACTION

The TYPE attribute specifies whether the field is single-choice, multiple-choice or immediate-action.

Choices in a single-choice selection field appear as radio buttons. Selecting one of the choices de-selects the previously selected choice in the selection field.

Choices in a multiple-choice selection field appear as check boxes. The user can select one or more of the check boxes, or not select any.

Choices in an immediate-action selection field appear as pushbuttons. The ACTION tag must be coded within the CHOICE tag for an immediate action to occur when the choice is selected.

PMTLOC = ABOVE | BEFORE
This attribute specifies whether the *field-prompt-text* (if specified) occurs above or in front of the selection field.

PMTWIDTH = n
The PMTWIDTH attribute specifies the prompt-width, in character units, of the prompt for the selection field. If any prompt is longer than this width, that prompt will be word-wrapped to fit on multiple lines. The minimum width value is 0 and the maximum is 128 character units. This value overrides the value on the DTACOL tag.

The compiler issues a warning message if the specified PMTWIDTH is zero and text is present.

SELWIDTH = n
The SELWIDTH attribute specifies the space to be taken up by the choices in the selection field. It is useful for getting the choices to occupy the same amount of space. The minimum width value is 1 character unit and the maximum is 128 character units. If the text is longer than the SELWIDTH, the text is truncated. If you do not specify the SELWIDTH attribute on the SELFLD tag, the SELWIDTH attribute on any surrounding DTACOL tag is used. If there is no surrounding DTACOL tag, the text of each choice determines the selection width, up to a maximum of 128.

The width of a selection field includes two parts: the check box or radio button, and the choice text. For immediate-action selection fields, it includes the text and the surrounding pushbutton border.

DIR = VERT | HORIZ
The DIR attribute specifies the direction in which the choices are arranged on the panel. The default direction is vertical.

If you code DIR = VERT, the choices are laid out top to bottom. If you code DIR = HORIZ, the choices are laid out left to right.

field-prompt-text
This allows you to specify the prompt text for the field. The prompt text can appear in front of or above the field, based on the value of the PMTLOC attribute. If you do not specify this text, no prompt text for this field appears on the panel.

Description

The SELFLD tag defines a field that includes a list of choices. CHOICE tags are used to identify each choice in a list.

To associate an action with the choice, you must code one or more ACTION tags within the start and end CHOICE tags.

Note: Control does not return automatically to the application unless the ACTION tags coded within the CHOICE tags specifies that control is to be returned to the application.

Conditions

If you want any action to occur, you must code an ACTION tag below the choice.

The selection field must contain choices for radio buttons, check boxes or pushbuttons to appear.

To conform to CUA and have the expected interaction:

- Specify the CHECKVAR attribute of the CHOICE tag.
- All choices within an immediate-action selection field or a single-choice selection field must name the same variable in the CHECKVAR attribute and have different values for the MATCH values.
- Choices within a multiple-choice selection field must name different variables in their CHECKVAR attributes.

Nested Tags

You can code the following tag within the SELFLD tag:

Tag	Dialog Element	Usage	Page
CHOICE (selection choice)	Application Panel	Multiple	15-28

Examples

The following markup shows how two selection fields can work together to dynamically de-emphasize choices in one selection field based on a selection made in another selection field. Several choices have already been selected.

The first selection field is single-choice, and allows the user to choose the vehicle type; either a sports car or a pick-up truck. When the sports car choice is selected by the user, the SETVAR action sets the variable *truck* to false and sets the variable *car* to true (the default value for the MATCH attribute). The variable *truck* and the variable *car* are used as the *availvars* on the options in the second selection field. *Truck* is the variable named as the *availvar* for the options: all terrain tires, 4-wheel drive, and bed-liner. By setting *truck* to false, the options specific only to trucks are de-emphasized. By setting the sports car choice to true, the options for the car that might have been previously de-emphasized are now selectable by the user.

Figure 15-49 on page 15-160 shows the formatted result.

```
<!doctype dm system>

<varclass name=sampcls type ='char 1'>
</varclass>

<varlist>
  <vardcl name=car     varclass=sampcls>
  <vardcl name=truck   varclass=sampcls>
  <vardcl name=ragtop  varclass=sampcls>
  <vardcl name=ac      varclass=sampcls>
  <vardcl name=leather varclass=sampcls>
  <vardcl name=fourwd  varclass=sampcls>
  <vardcl name=amfm    varclass=sampcls>
  <vardcl name=sunroof varclass=sampcls>
  <vardcl name=att     varclass=sampcls>
  <vardcl name=bedline varclass=sampcls>
</varlist>

<panel name=selfld1 width=30>Buy A Car!
  <selfld pmtwidth=18 pmtloc=above>Choose the type of
                                   vehicle you would like:
    <choice checkvar=car>Sports Car
      <action setvar=truck value=0>
    <choice checkvar=car match=0>Pick-up Truck
      <action setvar=truck>
  </selfld>

  <divider>
  <selfld type=multi pmtwidth=18 pmtloc=above>Options Available:
    <choice availvar=car checkvar=ragtop>Convertible (car only)
    <choice checkvar=ac>Air conditioning (both)
    <choice availvar=car checkvar=leather>Leather interior (car only)
    <choice availvar=truck checkvar=fourwd>4-wheel drive (truck only)
    <choice checkvar=amfm>AM/FM stereo (both)
    <choice availvar=car checkvar=sunroof>Sunroof (car only)
    <choice availvar=truck checkvar=att>All terrain tires (truck only)
    <choice availvar=truck checkvar=bedline>Bed-liner (truck only)
  </selfld>

</panel>
```

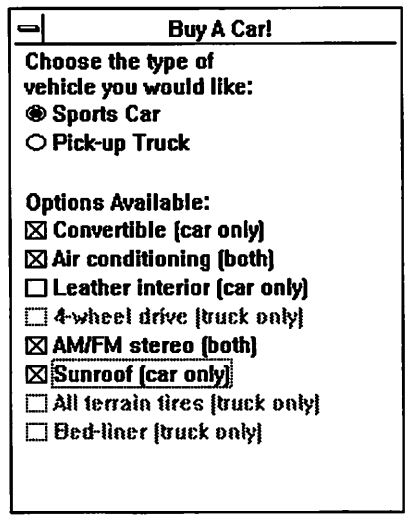
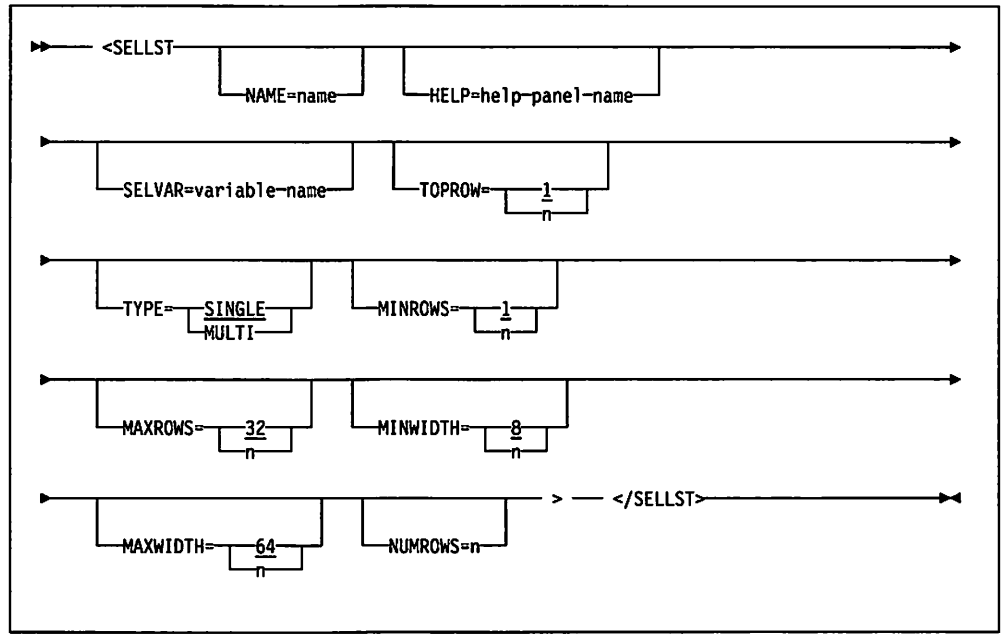


Figure 15-49. Single-Choice Selection Field

SELLST (Selection List)

The SELLST tag defines a scrollable list from which the user can select choices.



NAME = name

This attribute specifies the name of the selection list. This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the selection list.
- On an ADDPOP call to position a pop-up window relative to a selection list.
- On the DISPLAY call to place the cursor in the selection list.
- On the PANEL definition to place the cursor in the selection list.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name

This specifies the name of the help panel associated with the selection list. If the user selects help on a selection list, the help defined for the selection list displays. If no help is specified for the selection list, the extended help panel displays. If an extended help panel was not defined, a message is displayed stating that no help is defined. The *help-panel-name* must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

SELVAR = variable-name
This attribute specifies the name of the variable that indicates which items were preselected by the application and which items were selected. Do not use % notation.

Preselection is done by setting SELVAR to the index of the item or items that are to be preselected.

This variable must be a scalar variable if TYPE = SINGLE. Default selections are specified by setting the SELVAR to the index of the item that should be preselected. A zero indicates that no item is preselected. When control returns to the application after the display is complete, the SELVAR will have been set by Dialog Manager to the index of the item that was selected, or to zero if no items were selected by the user.

This variable must be an array variable if TYPE = MULTI and should have at least as many elements as are in the selection list. Preselection is done by setting the sequential elements (starting with the first) of the SELVAR array to the index of the item or items that should be preselected. When control returns to the application, sequential SELVAR array elements will have been set by Dialog Manager to the index or indexes of the item or items that were selected by the user. The sequence is ended by a zero in an array element or the end of the array, whichever occurs first. If the user did not make any selections, the first element of the SELVAR array will contain a zero.

TOPROW = 1 | n
The TOPROW attribute specifies the index of the selection list item that should appear at the top of the selection list when it is first displayed. This value can be a constant or a variable name. If this value is not specified, the first item in the selection list will appear at the top. The range is 1 to 1024. This value cannot be greater than NUMROWS.

The selection item specified with the TOPROW attribute will appear at the top of the selection list only if the selection list has enough room to scroll that item to the top without leaving blank items at the bottom of the selection list.

If a variable name is used, it must be coded using % notation.

TYPE = SINGLE|MULTI
The TYPE attribute specifies the type of the selection list. In a single-choice selection list, the user can only select one item from the list. If the type is multiple-choice, the user can select more than one item. The default is single-choice.

MINROWS = 1|n
The MINROWS attribute specifies the minimum number of rows that SELLST can be sized to. The range is 0 to 512.

MAXROWS = 32|n
The MAXROWS attribute specifies the maximum number of rows that SELLST can be sized to. The range is 1 to 512.

MINWIDTH = 8|n
The MINWIDTH attribute specifies the minimum width, in character units, that SELLST can be sized to. The range is 1 to 512.

MAXWIDTH = 64|n
The MAXWIDTH attribute specifies the maximum width, in character units, that SELLST can be sized to. The range is 1 to 512.

NUMROWS = n

The NUMROWS attribute specifies the number of items in the selection list. To view all of the items, the user can scroll the selection list. The NUMROWS value can be a constant or a variable.

If the value is not specified, the Dialog Manager assumes that the number of rows is the number of elements of the smallest array variable specified for DATAVAR (SELCOL tag). The range is 1 to 1024. If NUMROWS is a variable, it can have a value of 0 to 1024 at run time.

If a variable name is used, it must be coded using % notation. (For example, to code the variable *numitems*, code '%numitems'.)

Description

The SELLST tag defines a scrollable list from which the user can select choices. The items displayed in the selection list are defined by the array variable on the SELCOL tag.

If you want a fixed width always visible for your selection list, specify the same value for MINWIDTH and MAXWIDTH. If you want a fixed number of rows to always be visible, specify the same value for MINROWS and MAXROWS. If the same value is specified for MINROWS, MAXROWS and NUMROWS is less than this value, blank space will appear in the selection list.

Conditions

You must code only one SELCOL tag within this tag.

Nested Tags

You can code the following tags within the SELLST tag:

Tag	Dialog Element	Usage	Page
SELCOL (selection column)	Application Panel	Single	15-154

Example

The following markup shows a multiple-choice selection list. The number of items in the selection list is defined in the variable, *numfiles*. The MAXROWS attribute is used to control the space occupied by the selection list when the window is sized. The maximum number of items that will be displayed in the selection list is 15. The data displayed in the selection list comes from the array data variable, *filenms*, and the array selection variable, *cfiles*, is updated to indicate which files the user selected. Figure 15-50 shows the formatted result.

```
<!doctype dm system>

<varclass name=namecls type='char 15'>
<varclass name=numcls type='numeric 5'>

<varlist>
  <vardcl name=filenms  varclass=namecls maxdim=200>
  <vardcl name=cfiles   varclass=numcls  maxdim=200>
  <vardcl name=numfiles varclass=numcls>
</varlist>


<panel name=sellst1>Copy Files
  <topinst>Select the files to be copied.
  <sellst selvar=cfiles type=multi numrows='%numfiles' maxrows=15>
    <selcol datavar=filenms>
  </sellst>
</panel>
```

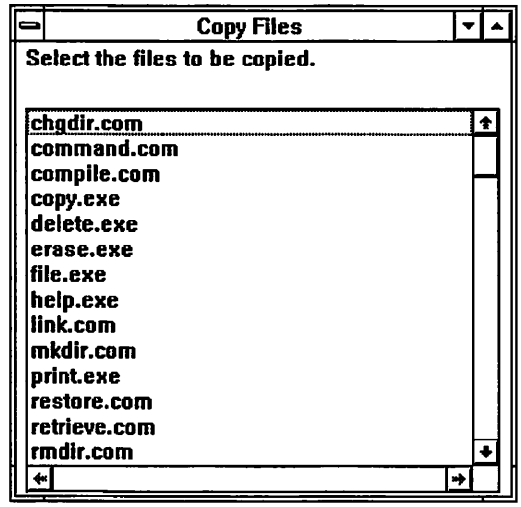
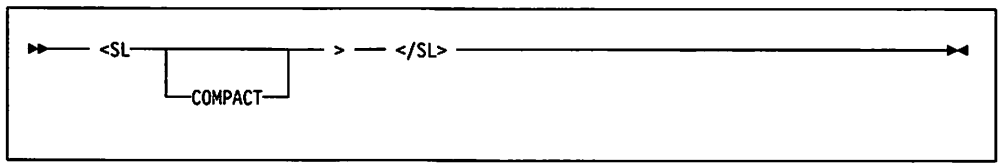


Figure 15-50. Selection List

SL (Simple List)

The SL tag defines a simple list of items.



COMPACT
The COMPACT attribute specifies that the list formats without blank lines between the items.

Description

The SL tag defines a simple list of items. Simple lists are indented lists, with no preceding item identifiers. Nested lists (lists embedded within other lists) indent four spaces to the right of the left margin of the list that contains them. The compiler adds a blank line before the first item in the list.

You use the LI (list item) to denote each list item. See "LI (List Item)" on page 15-92 for additional information on how to define simple lists.

Conditions

You can use the SL tag only within an information region.

Nested Tags

You can code the following tags within the SL tag:

Tag	Dialog Element	Usage	Page
LI (list item)	Application Panel Help Panel	Multiple	15-92
LP (list part)	Application Panel Help Panel	Multiple	15-99

Example

The following application panel markup contains both a simple list and compact simple list. Figure 15-51 on page 15-166 shows the formatted result.

```
<!doctype dm system>

<panel name=sl1 depth=13 width=70>Automatic teller
  <region dir=horiz>
  <info width=38>
    <p>To access this automatic teller you must:
    <sl>
      <li>Insert Cash Card into slot.
      <sl compact>
        <li>Turn the card so that the magnetic strip faces up.
        <li>Slide the card in until you feel a slight resistance.
      </sl>
      <li>Enter your password.
      <li>Choose a transaction.
    </sl>
  </info>
  <divider type=solid gutter=9>
  <selfld type=single>Choose a transaction
    <choice>Transfer
    <choice>Make a deposit
    <choice>Account balance
    <choice>Withdraw cash
  </selfld>
  </region>

</panel>
```

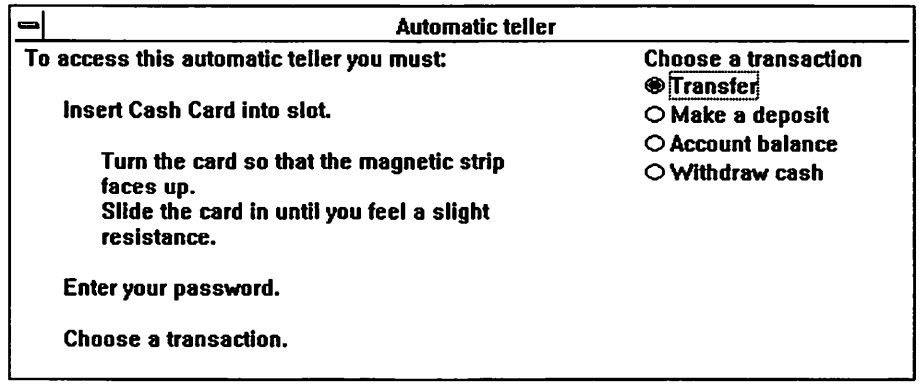
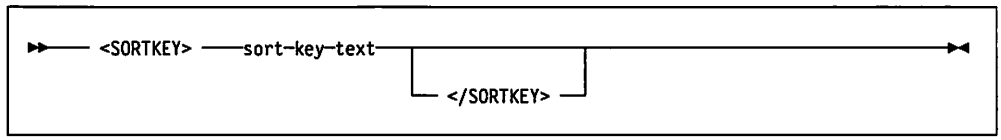


Figure 15-51. Simple List

SORTKEY (Sort Key)

The SORTKEY tag defines a sort key used instead of the actual command or index topic text when creating an index.



sort-key-text
Sort-key-text specifies the text of the sort key. The maximum length of the text, including embedded blanks, is 64 bytes.

Description

The SORTKEY tag defines a sort key used instead of the actual command or index topic text when creating an index.

The sort key applies to the contents of the ICMD and the ITOP tags. It must be coded following the *external-command-string* within an ICMD (index command).

In the ITOP (index topic) tag, the sort key must be coded following the *index-topic-text*. The sorting only affects the way in which the ICMD and ITOP data is displayed at run time when command help or help index is requested.

You use the SORTKEY tag to override the default ordering of commands and topics. For example, to group the following commands, give them the same sort key: COPY, DELETE, RENAME. Then, put more commands into a second group: FORMAT, PRINT, SEND. Here is how they will group with and without the SORTKEY tags:

```
With SORTKEY          Without SORTKEY

 COPY                  COPY
 DELETE                DELETE
 RENAME                FORMAT
 FORMAT                PRINT
 PRINT                 RENAME
 SEND                  SEND
```

Conditions

You can only code the SORTKEY tag within the ICMD and ITOP tags.

Note: This tag will be supported when full DBCS support is available in *OS/2 Programming Tools and Information.*

Nested Tags

None.

Example

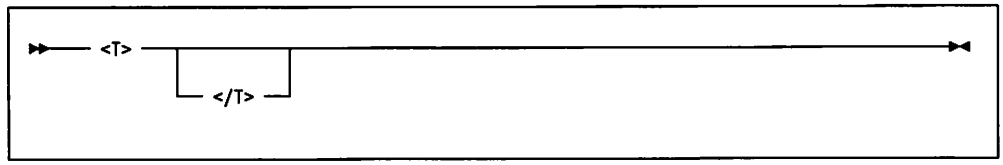
The help panel examples contain an index topic definition with a specified sort key string.

```
<help name=sortke1 depth=20 width=35>Help for Copying a File
<itop roots=copy>Copying<sortkey>Filing
  <area>
  <info>
    <p>You need specify the following items
    to copy a file.
    <ul>
      <li>File name
      <li>Number of copies
      <p>If you do not specify a number, the
      default is 1.
      <li>Bin number the copy or copies are to be sent to.
    </ul>
  </info>
  </area>
</help>


<help name=sortke2 depth=20 width=35>Help for Renaming a File
<itop roots=rename>Renaming<sortkey>Filing
  <area>
  <info>
    <p>You need specify the following items
    to Rename a file.
    <ul>
      <li>Current name of the file
      <li>New name for the file
    </ul>
  </info>
  </area>
</help>
```

T (Truncation)

The T tag designates the minimum command name that the user must enter to issue a command.



Description

The T tag designates the minimum command name that the user must enter to issue a command. You code the T tag within the *external-command-name* on the CMD tag. For example, if the following command is coded in an application command table, the user could enter "com," "comp," "compa," "compar," or "compare," to issue the command.

```
<cmd name=compl>com<t>pare
```

The minimum command name must be at least two bytes in length. For example, you cannot use "C" for copy.

You should also be careful to avoid specifying values that conflict with other commands. For example:

```
<cmd name=compl>co<t>mpare
<cmdact action=passthru>

<cmd name=copy>co<t>py
```

would cause the compiler to issue a warning, because "co" can resolve to two different commands. At run time, the Dialog Manager runs the first valid command in the command table that matches the command issued—in this case, the COMPARE command.

Omitting the T tag is the same as specifying the tag at the end of the *external-command-name*.

Both of the following examples indicate that the user must enter the entire command name to run the command.

```
<cmd name=rename>rename
<cmd name=rename>rename<t>
```

Note: While a matching truncation end tag is allowed within the command text, its placement is ignored by the compiler.

Conditions

This tag is allowed only within a CMD tag.

Nested Tags

None.

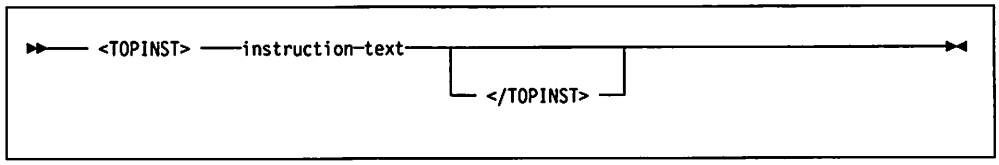
Example

Each of the commands within this command table have a truncation definition. The minimum command for **Open** is "Op." The minimum command for **Close** is "Cl." The minimum command for **Copy** is "Co."

```
<cmdtbl applid=demo>
  <cmd name=open>Op<t>en
    <cmdact action=passthru>
  <cmd name=close>Cl<t>ose
    <cmdact action=passthru>
  <cmd name=view>Co<t>py
    <cmdact action=passthru>
</cmdtbl>
```

TOPINST (Top Instruction)

The TOPINST tag defines a top instruction for a panel.



instruction-text
Instruction-text specifies the text to appear as the top instruction in a panel.

Description

The TOPINST tag defines a top instruction for a panel. The text is an implied paragraph. The top instructions are formatted (word-wrapped) based on the width specified on the PANEL tag or a width of 128, whichever is smaller, and will not reformat when the window is sized.

Conditions

You must specify the top instructions, if any, as the first items in a panel body definition. That is, following the action bar definition and preceding the AREA tag or any tags allowed within AREA.

Code multiple TOPINST tags within a PANEL definition to define multiple instructions.

Top instructions are not scrollable.

Nested Tags

None.

Example

The following application panel markup contains a top instruction. Figure 15-52 on page 15-172 shows the formatted result.

```
<!doctype dm system>

<varclass name=numcls   type = 'char 5'>
<varclass name=sampcls  type = 'char 20'>
<varclass name=statcls  type = 'char 2'>

<varlist>
  <vardcl name=num      varclass=numcls>
  <vardcl name=name     varclass=sampcls>
  <vardcl name=addr     varclass=sampcls>
  <vardcl name=city     varclass=sampcls>
  <vardcl name=stat     varclass=statcls>
  <vardcl name=zip      varclass=numcls>
</varlist>

<panel name=topins1 depth=15 width=60>Wholesale Food Distributors
  <topinst>Complete the following information, and press Enter.
  <area>
    <dtacol pmtwidth=12 entwidth=10>
      <dtafld datavar=num>Customer number
      <dtafld datavar=name>Name
      <dtafld datavar=addr>Address
    </dtacol>
    <region dir=horiz>
      <dtafld datavar=city>City
      <dtafld datavar=stat>State
      <dtafld datavar=zip>Zip code
    </region>
  </area>
</panel>
```

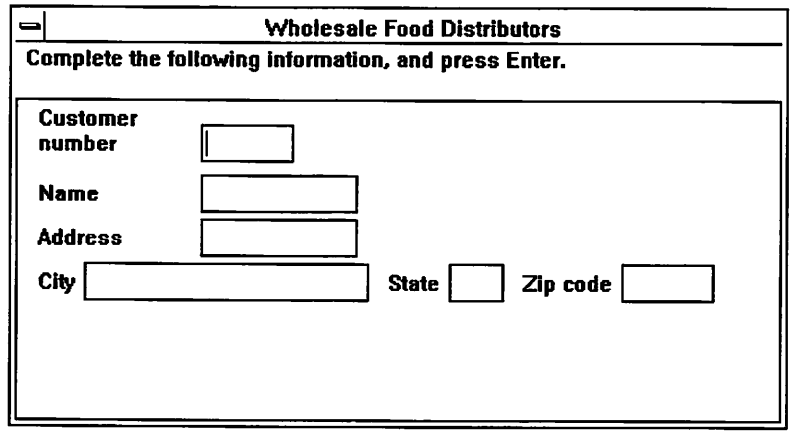
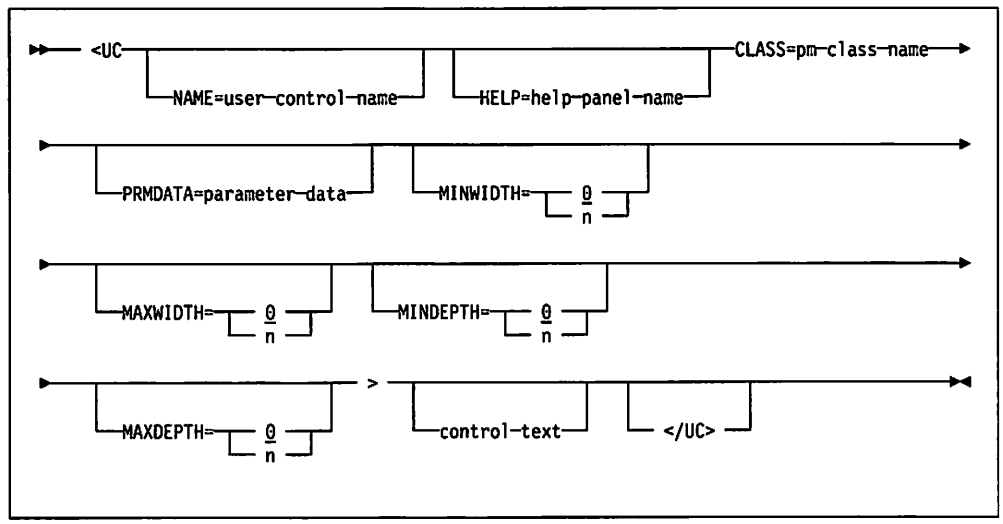


Figure 15-52. Top Instruction

UC (User Control)

The UC tag allows you to specify an application-defined control on a panel.



NAME = user-control-name
This specifies the name of the user control.

This name can be used:

- On the DISPLAY call to position a message pop-up window relative to the user control
- On the ADDPOP call to position a pop-up window relative to the user control
- On the DISPLAY call to place the cursor as defined by the user control application.
- On the PANEL definition to place the cursor.

The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

HELP = help-panel-name
This is the name of a help panel specified on a HELP tag. It identifies the help panel to be displayed when help about this control is chosen by the user. If no help is specified for the user control, the extended help panel is displayed. If an extended help panel was not defined, a message is displayed stating that no help is defined. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

CLASS = pm-class-name
This specifies the name of the window class to which the user control belongs.

PRMDATA = parameter-data
The PRMDATA attribute contains any specific parameters needed by the user control. The control receives this data unchanged at run time, allowing configuration data to be used if the author of the control needs it. This string resides in the non-translatable section of the tag, making it useful for non-translatable material. You can specify the name of a dialog variable (using the % notation) whose value at run time will be passed as the parameter-data. To specify a string that begins with a %, you must code an additional % before the string to distinguish it from a variable name. (For example, to specify the string %abc, code '%%abc'; to specify variable abc, code '%abc'.)

The length of this attribute is limited to 253 bytes.

MINWIDTH = 0 | n
The MINWIDTH attribute specifies the minimum width, in character units, that the user control can be sized to. The range is 0 to 512.

MAXWIDTH = 0 | n
The MAXWIDTH attribute specifies the maximum width, in character units, that the user control can be sized to. The range is 0 to 512.

MINDEPTH = 0 | n
The MINDEPTH attribute specifies the minimum depth, in character units, that the user control can be sized to. The range is 0 to 512.

MAXDEPTH = 0 | n
The MAXDEPTH attribute specifies the maximum depth, in character units, that the user control can be sized to. The range is 0 to 512.

control-text
The *control-text* specifies the text of the user control. The window procedure of the control can use it. Typically, this is for displayed strings that are used when the control is displayed. Unlike PRMDATA, this field is translatable and should be used when translatable material is handled by the control.

Description

The UC tag allows you to specify an application-defined control on a panel. The user interface found in Presentation Manager on OS/2 systems is constructed of various types of components called controls. When an application needs to construct a panel that is made up of something other than the Dialog Manager fields, you must write a new class. The code for this new class is registered with Presentation Manager at run time. These new fields are called *user controls*. For more information, refer to the additional features of the Dialog Manager chapter in the *Dialog Manager Guide and Reference*.

Conditions

You can code this tag only within AREA, PANEL and REGION. You cannot code this tag on a help panel.

Nested Tags

None.

Example

The following application panel markup specifies a user control displayed in a pop-up window. The user control allows the user to verify an order by signing the panel.

```
<panel name=uc1 width=42 depth=11>Signature
  <region dir=vert>
    <divider type=none gutter=2>
    <region dir=horiz>
      <divider type=none gutter=2>
      <uc class=Signature>
      </uc>
    </region>
    <divider type=none gutter=1>
    <selfld help=KEYSHELP type=action dir=horiz>
      <choice checkvar=fkaType match=0><m>Accept
        <action setvar=fkaType value=0>
        <action run=ENTER>
      <choice checkvar=fkaType match=1><m>Cancel
        <action setvar=fkaType value=1>
        <action run=CANCEL>
      <choice><m>Help
        <action run=EXHELP>
    </selfld>
  </region>
</panel>
```

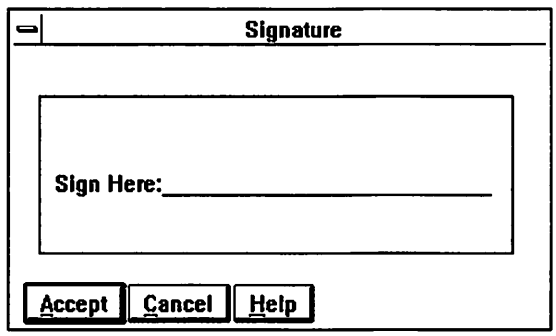
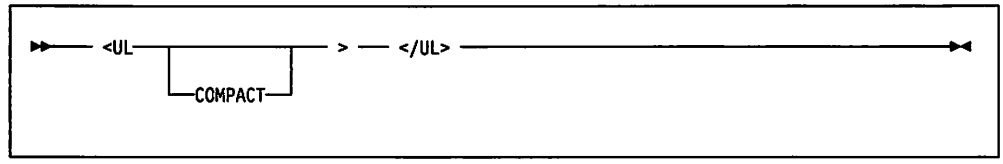


Figure 15-53. User Control

UL (Unordered List)

The UL tag defines an unordered list.



COMPACT
The COMPACT attribute causes the list to format without a blank line between the items.

Description

The UL tag defines an unordered list. The list will appear indented with a list item identifier in the left margin. Nested lists (lists embedded within other lists) indent four spaces to the right of the left margin of the list that contains them. A blank line is added before the first item in the list.

There are three levels of item identifiers; bullets (lowercase o), hyphens (-), and dashes (—). Each level of identifier is used successively when you nest unordered lists.

Note: The representations of the level indicators are tailorable by language. For more information, see "DTLC" on page 13-2.

You use the LI (list item) to denote each list item. See "LI (List Item)" on page 15-92 for additional information on how to define unordered lists.

Conditions

You can use this tag only within an information region.

Nested Tags

You can code the following tags within the UL tag:

Tag	Dialog Element	Usage	Page
LI (list item)	Application Panel Help Panel	Multiple	15-92
LP (list part)	Application Panel Help Panel	Multiple	15-99

Example

The following help panel markup contains an unordered list. The second list item contains a compact unordered list. Figure 15-54 shows the formatted result.

```
<help name=ul1>Help for the Hardware Department
  <area>
  <info>
    <p>The hardware department carries the following tools:
    <ul>
      <li>Hand tools
      <li>Power tools
      <ul compact>
        <li>Drills
        <li>Saws
        <li>Routers
        <li>Sanders
      </ul>
      <li>Bench tools
      <li>Garden implements
    </ul>
  </info>
  </area>
</help>
```

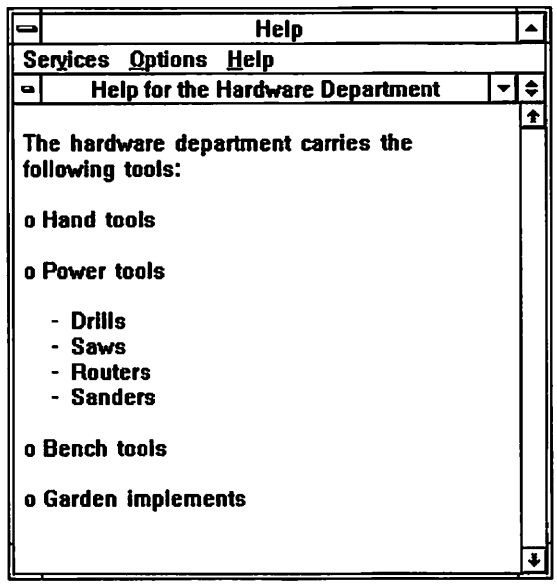
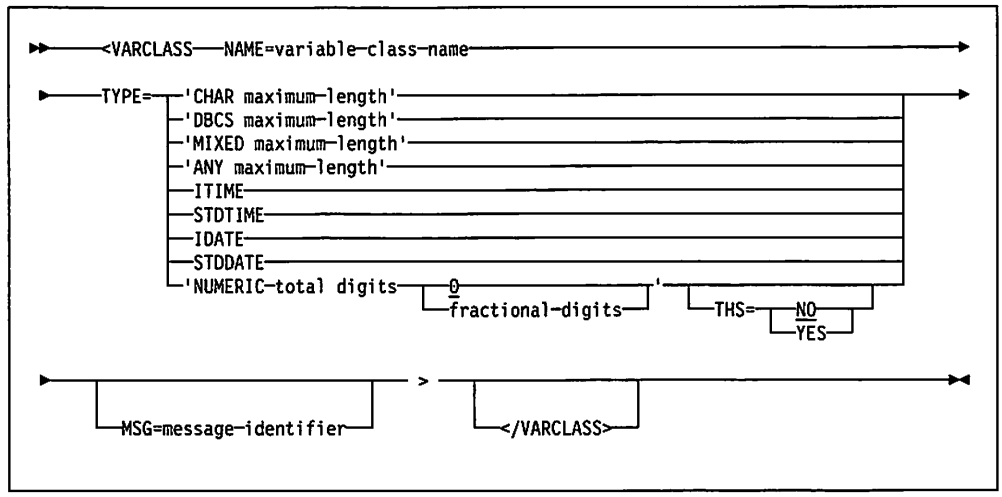


Figure 15-54. Unordered List

VARCLASS (Variable Class)

The VARCLASS tag defines variable class related information.



NAME = variable-class-name

The NAME attribute specifies the name of this variable class. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

Lowercase characters are translated to their uppercase equivalents.

TYPE = type of data

The TYPE attribute specifies the format of the data received from the Dialog Manager variable pools or from a Presentation Manager class (as determined by using VARLIST).

'CHAR maximum-length'

This specifies a character string for which the maximum length, in bytes, is given by *maximum-length.* CHAR accepts only single-byte character strings. The maximum internal and external lengths of a dialog variable 32767.

'DBCS maximum-length'

This is a double-byte character string for which the maximum length, in bytes, is given by the maximum length attribute. The maximum internal and external lengths of a dialog variable are 32766. The default pad character is the double-byte space character. The maximum length must be an even number.

Note: This is permitted only when the code page specified on the DTLC command of the compiler supports double-byte characters. Otherwise, an error message is issued by the compiler.

'MIXED maximum-length'

This specifies a character string containing single-byte characters, double-byte characters, or both for which the maximum length, in bytes, is given by the maximum length attribute. The maximum internal and external lengths of a data type are 32767. The maximum length allowed is reduced by two bytes whenever there is a transition from single-byte to

double-byte characters. The default pad character is the single-byte character space. This type is used for portability to other environments since its behaviour is compatible with EBCDIC DBCS. This type is treated as CHAR if the system does not support double-byte characters.

Note: If this attribute is used when the code page specified does not support double-byte characters, it is treated as if CHAR is the TYPE. This TYPE is used for poratbility to other environments.

'ANY maximum-length'
This specifies a character string containing single-byte characters, double-byte characters or both whose maximum length, in bytes, is given by maximum length. The maximum internal and external lengths of a data type are 32767. The default pad character is the single-byte character space.

You can use this attribute only in OS/2 applications

Note: If this used when the code page specified does not support double-byte characters, it is treated as if CHAR is the TYPE. This can be used to enable your application to support both single-byte and double-byte character systems.

ITIME
This specifies a data type for which the format represents time expressed in hours (HH) and minutes (MM).

The ITIME internal format used by the dialog variable contains four digits representing HHMM. The ITIME display format contains five characters including the national language time separator character (ZDATEF). For the United States, the format is HH:MM. Hours are specified using the 24 hour clock.

On output, the translate lists defined for the particular VARCLASS are applied to the value of the dialog variable. If an explicit match is not found or if no translate lists are defined, the value of the dialog variable is automatically converted from ITIME internal format to ITIME display format. In this case, for example, a dialog variable with the value 0345 is displayed as 03:45 (for the U.S.). See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

On input, the translate lists defined for the particular VARCLASS is applied to the value entered by the user. If an explicit match is not found or if no translate lists are defined, the value entered is automatically converted from ITIME display format to ITIME internal format. The conversion assumes the value entered is in the form HH*s*MM, where *s* is the time separator character as specified by ZDATEF. (You should display the expected user format [perhaps as a data description] to ensure that the characters entered will translate properly to the expected internal format). In the U.S., for example, a displayed value of 03:45 is stored in the dialog variable as 0345.

For input only, the Dialog Manager ensures the resulting ITIME internal value is a valid time. It ensures that the internal value for HH is 00 – 23, and for MM is 00 – 59.

When an error is found on input, and the user has not selected Cancel or Exit, Dialog Manager displays one of three default messages: One for a time that is not valid (for example, 87:00), one for non-numeric characters specified in the hours or minutes part of the time, and one if the time delimiter is not the time delimiter specified as the national language time

separator (ZDATEF). You can override the DM-provided messages using the MSG attribute. If no errors are found, any check lists defined for the VARCLASS are applied.

Note: To specify the current time (hours and minutes), use the system variable ZITIME in panel definitions.

STDTIME

This specifies a data type for which the format represents time expressed in hours (HH), minutes (MM) and seconds (SS).

The STDTIME internal format used by the dialog variable contains six digits representing HHMMSS. The STDTIME display format contains eight characters including the national language time separator character (ZDATEF). For the United States, the format is HH:MM:SS. Hours are specified using the 24 hour clock.

On output, the translate lists defined for the particular VARCLASS are applied to the value of the dialog variable. If an explicit match is not found or if no translate lists are defined, the value of the dialog variable is automatically converted from STDTIME internal format to STDTIME display format. For example, a dialog variable with the value 034555 is displayed as 03:45:55 (for the U.S.). See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

On input, the translate lists defined for the particular VARCLASS is applied to the value entered by the user. If an explicit match is not found or if no translate lists are defined, the value entered is automatically converted from STDTIME display format to STDTIME internal format. The conversion assumes the value entered is in the form HH*s*MM*s*SS, where *s* is the time separator character as specified by ZDATEF. (You should display the expected user format [perhaps as a data description] to ensure that the characters entered will translate properly to the expected internal format). In the U.S., for example, a displayed value of 03:45:55 is stored in the dialog variable as 034555.

For input only, the Dialog Manager ensures the resulting STDTIME internal value is a valid time. It ensures that the internal value for HH is 00 – 23, for MM is 00 – 59 and for SS is 00 – 59.

When an error is found on input and the user has not selected Cancel or Exit, Dialog Manager displays one of three default messages: One for a STDTIME that is not valid (for example, 12:87:00), one for non-numeric characters specified in the hours, minutes, or seconds part of the STDTIME, and one if the time delimiter is not the time delimiter specified as the national language time separator (ZDATEF). You can override the DM-provided messages using the MSG attribute. If no errors are found, any check lists defined for the VARCLASS are applied.

Note: To specify the current time (hours, minutes and seconds), use the system variable ZSTDTIME in panel definitions.

IDATE

This specifies a data type for which the format represents a date expressed in a two-digit year (YY), month (MM) and day (DD).

The IDATE internal format used by the dialog variable contains six digits representing YYMMDD. The IDATE display format contains eight characters including the national language date format (ZDATEF). For the United States, the format is MM-DD-YY.

On output, the translate lists defined for the particular VARCLASS are applied to the value of the dialog variable. If an explicit match is not found or if no translate lists are defined, the value of the dialog variable is automatically converted from IDATE internal format to IDATE display format. In this case, for example, a dialog variable with the value 880701 is displayed as 07-01-88 (for the U.S.). See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

On input, the translate lists defined for the particular VARCLASS is applied to the value entered by the user. If an explicit match is not found or if no translate lists are defined, the value entered is automatically converted from IDATE display format to IDATE internal format. The conversion assumes the value entered is in the national language format and uses the date separator character as specified by ZDATEF. (You should display the expected user format [perhaps as a data description] to ensure that the characters entered will translate properly to the expected internal format). For example, a displayed value of 07-01-88 in the U.S. is stored in the dialog variable as 880701.

For input only, the Dialog Manager ensures the resulting IDATE internal value is a valid date. It ensures that the internal value for YY is 00-99, for MM is 01 – 12 and for DD is 01 – 31. Validation is also done to check the date for months with fewer than 31 days and for leap years.

When an error is found on input and the user has not selected Cancel or Exit, Dialog Manager displays one of three default messages: One for an IDATE that is not valid (for example, 04-32-88), one for non-numeric characters specified in the year month or day part of the IDATE, and one if the date delimiter is not the date delimiter specified as the national language date format (ZDATEF). You can override the DM-provided messages using the MSG attribute. If no errors are found, any check lists defined for the VARCLASS are applied.

Note: To specify the current date (two-digit year, month and day), use the system variable ZIDATE in panel definitions.

STDDATE

This specifies a data type for which the format represents a date expressed in a four-digit year (YYYY), month (MM) and day (DD).

The STDDATE internal format used by the dialog variable contains eight digits representing YYYYMMDD. The STDDATE display format contains ten characters including the national language date separator character (ZDTS). For the United States, the format is MM-DD-YYYY.

On output, the translate lists defined for the particular VARCLASS are applied to the value of the dialog variable. If an explicit match is not found or if no translate lists are defined, the value of the dialog variable is automatically converted from STDDATE internal format to STDDATE display format. For example, a dialog variable with the value 19880701 is displayed as 07-01-1988 (for the U.S.). See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

On input, the translate lists defined for the particular VARCLASS is applied to the value entered by the user. If an explicit match is not found or if no translate lists are defined, the value entered is automatically converted from STDDATE display format to STDDATE internal format. The conversion assumes the value entered is in the national language format and uses the national language date separator character as specified by ZDATEF. (You should display the expected user format [perhaps as a data

description] to ensure that the characters entered will translate properly to the expected internal format). For example, a displayed value of 07-01-1988 in the U.S. is stored in the dialog variable as 19880701.

For input only, the Dialog Manager ensures the resulting STDDATE internal value is a valid date. It ensures that the internal value for YYYY is 0000-9999, for MM is 01−12 and for DD is 01−31. Validation is also done for months with fewer than 31 days and for leap years.

When an error is found on input and the user has not selected Cancel or Exit, Dialog Manager displays one of three default messages: One for a STDDATE that is not valid (for example, 04-32-1988), one for non-numeric characters specified in the year, month or day part of the STDDATE, and one if the date delimiter is not the date delimiter specified as the national language date separator (ZDATEF). You can override the provided messages using the MSG attribute. If no errors are found, any check lists defined for the VARCLASS are applied.

Note: To specify the current date (four-digit year, month and day), use the system variable ZSTDDATE in panel definitions.

'NUMERIC total digits fractional digits'

This specifies a data type whose format represents numeric data.

The numeric internal format used by the dialog variable is always the same and can be described as follows:

```
[±] [..d] [p[d.. ]]
```

where *d* is a digit between (0−9) and *p* is the decimal separator which is always a period (.). If fractional digits are omitted, it defaults to 0. If a number is positive, it may or may not have a sign. The numeric display format is dependent on the national language format. The decimal separator and sign are replaced by the national language decimal separator and national language sign characters. The sign character is inserted in the appropriate place in the number as required by the display format. Space is reserved for national language thousands separators when THS=YES.

On output, the translate lists defined for the particular VARCLASS are applied to the value of the dialog variable. If an explicit match is not found or if no translate lists are defined, the value of the dialog variable is automatically converted from numeric internal format to numeric display format. In this case for example, a dialog variable with the value -1234.56 is displayed as -1,234.56 (for US, and THS=YES). See "XLATL (Translate List)" on page 15-197 for more information on explicit matches.

On input, the translate lists defined for the particular variable class is applied to the value entered by the user. If an explicit match is not found or if no translate lists are defined, the value entered is automatically converted from numeric display format to numeric internal format. The conversion assumes the value entered is in the national language numeric format. The user always has the option of entering the thousands separators. However, if THS=NO, space is not reserved for the separators. If the user enters separators, there may not be enough space reserved for numeric digits. In this case for example, a displayed value of 1,234.56 is stored in the dialog variable as 1234.56.

For input only, the Dialog Manager ensures the resulting numeric internal value is valid. When an error is found on input and the user has not selected Cancel or Exit, Dialog Manager displays one of the default messages depending on the error:

- Incorrect sign
- Incorrect decimal separator
- Incorrect thousands separator
- Non-numeric characters
- Incorrect number of total digits or fractional digits.

The total number of digits allowable on NUMERIC type is 31.

THS = NO | YES
This attribute specifies if thousands separators are used when displaying NUMERIC values. This attribute is only valid when TYPE = NUMERIC is specified.

MSG = message-identifier
The MSG attribute indicates the message to be displayed if the variable fails the validity check implied by the TYPE specification.

The Dialog Manager provides a default message if you do not specify one on the MSG attribute.

Note: If Exit or Cancel is selected an error is returned to the application when input translations fail. For more information, refer to the field validation section of the *Dialog Manager Guide and Reference*.

Description

The VARCLASS tag defines variable class related information. You code the VARCLASS tag within the same source file as the referencing CMDTBL tag, MSGMBR tag, or PANEL tags.

Note: If the variable value is not displayed, the checks, edits, and translations specified by that VARCLASS are not used.

Conditions

You must code all VARCLASS tags before any other tags.

Nested Tags

You can code the following tags within the VARCLASS tag.

Tag	Dialog Element	Usage	Page
CHECKL (check list)	None	Multiple	15-26
XLATL (translate list)	None	Multiple	15-197

Example

The following markup contains a variable class that defines translation lists and validity checks. The markup also contains five data fields that reference variable classes.

```
<!doctype dm system>
<varclass name=namec type='CHAR 25' >
   <checkl>
      <checki type=alpha>
   </checkl>
</varclass>

<varclass name=officec type='CHAR 4'>
   <xlatl format=upper>
   </xlatl>
   <checkl>
      <checki type=chars parm1=EQ parm2='ABCDEF0123456789'>
   </checkl>
</varclass>
```

```
<varclass name=deptc type='CHAR 16'>
  <xlatl format=upper>
  </xlatl>
  <xlatl format=none>
     <xlati value=CS>Computer Science
     <xlati value=EE>Electrical Engineer
     <xlati value=CE>Chemical Engineer
  </xlatl>
</varclass>

<varclass name=itimec type=ITIME>
   <checkl>
      <checki type=range parm1=0800 parm2=1700>
   </checkl>
</varclass>

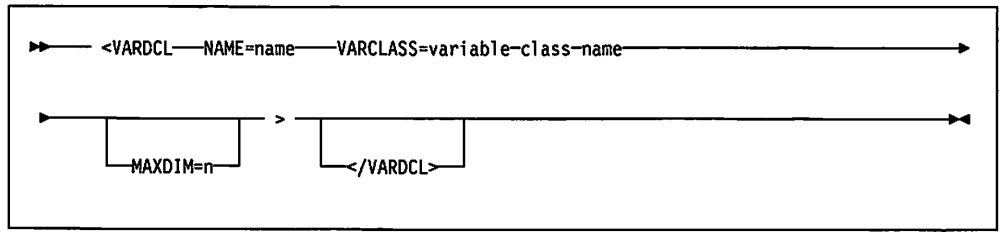
<varclass name=stddatec type=STDDATE>

<varlist>
   <vardcl name=ltime varclass=itimec>
   <vardcl name=ldate varclass=stddatec>
   <vardcl name=reserver varclass=namec>
   <vardcl name=office varclass=officec>
   <vardcl name=dept varclass=deptc>
</varlist>

<panel name=vrcls02>Conference Room
   <topinst>Enter the required information to reserve a conference room.
   <area>
     <dtacol pmtwidth=20 entwidth=10 Deswidth=40>
        <dtafld name=field1 datavar=reserver>Name
        <dtafld name=field2 datavar=office>Office number
        <dtafld name=field3 datavar=dept>Department name
        <dtafld name=field4 datavar=ltime>Time
        <dtafld name=field5 datavar=ldate>Date
     </dtacol>
     </area>
</panel>
```

VARDCL (Variable Declaration)

The VARDCL tag defines variables referenced in dialog elements.



NAME = name
The NAME attribute specifies the name of this variable used elsewhere on the panel. The name must have the following characteristics:

- 1 – 8 characters in length.
- The first (or only) character must be A – Z or a – z.
- Remaining characters, if any, must be A – Z, a – z, or 0 – 9.

VARCLASS = variable-class-name
The VARCLASS attribute specifies the default VARCLASS associated with this variable. If you want to perform a different set of checks or translations the DTAFLD, LSTCOL, SELCOL, or VARSUB tags you can specify an overriding VARCLASS of the same type.

MAXDIM = n
The MAXDIM attribute specifies the maximum number of entries when the variable is an array. This attribute allows a range of values from 1 – 65535. If coded, the value must be positive. If not coded, the variable is assumed to be scalar. (The maximum length specified in the associated variable class applies to each entry in the array.)

Description

The VARDCL tag defines variables referenced in dialog elements.

Conditions

You must code the VARDCL tag within a VARLIST tag.

You must declare all variables referenced by dialog elements.

Nested Tags

None.

Example

The following markup contains four array variable and three scalar variable declarations. The markup also contains a selection field and list field that reference the variables.

```
<!doctype dm system>

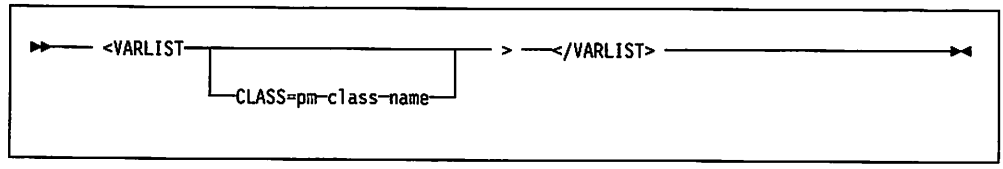
<varclass name=vflag type='CHAR 1' >
<varclass name=vday type='CHAR 1'>
   <xlatl>
      <xlati value=1>Monday
      <xlati value=2>Tuesday
      <xlati value=3>Wednesday
      <xlati value=4>Thursday
      <xlati value=5>Friday
   </xlatl>
<varclass name=vtime type=ITIME>
<varclass name=vdate type=STDDATE>
<varclass name=vseminar type='CHAR 1'>
   <xlatl>
      <xlati value=1>Public speaking
      <xlati value=2>Technical writing
      <xlati value=3>Personal finance
   </xlatl>

<varlist>
   <vardcl name=seminar varclass=vseminar Maxdim=5>
   <vardcl name=day varclass=vday Maxdim=5>
   <vardcl name=date varclass=vdate Maxdim=5>
   <vardcl name=time varclass=vtime Maxdim=5>
   <vardcl name=pubspk varclass=vflag>
   <vardcl name=techw varclass=vflag>
   <vardcl name=finan varclass=vflag>
</varlist>

<panel name=vrldcl>Weekly Seminar Schedule
   <area>
     <lstfld toprow=1 numrows=5>
        <lstcol datavar=seminar usage=out>Seminar
        <lstcol datavar=day usage=out>Day
        <lstcol datavar=date usage=out>Date
        <lstcol datavar=time usage=out>Time
     </lstfld>
     <selfld type=multi>Check the seminars you want descriptions for:
        <choice checkvar=pubspk><m>Public speaking
        <choice checkvar=techw><m>Technical writing
        <choice checkvar=finan>Personal <m>finance
     </selfld>
   </area>
</panel>
```

VARLIST (Variable List)

The VARLIST tag collects multiple variable declarations.



CLASS = pm-class-name
The CLASS attribute specifies the name of the access method for the variables in the list. This overrides the default access method that uses variable pools. For more information, refer to the user interaction chapter of the *Dialog Manager Guide and Reference*.

Description

The VARLIST tag collects multiple variable declarations. Information relative to the particular variables is stated on the VARDCL tags enclosed within the VARLIST tag. Every variable referenced must be declared in a VARLIST.

Multiple VARLISTs are allowed. However, there can be only one variable list that uses the Dialog Manager variable pool method to access variables. That is, you may have one variable list that does not use the CLASS attribute and one for each uniquely named variable access user exit. Dialog elements can reference variables from multiple variable lists.

Conditions

You code the VARLIST tags after any VARCLASS tags.

Nested Tags

You can code the following tag within the VARLIST tag:

Tag	Dialog Element	Usage	Page
VARDCL (variable declaration)	None	Multiple	15-186

Example

The following markup contains a variable list containing three variable declarations. The application panel in the markup contains a selection field and two data fields that reference the variable declarations.

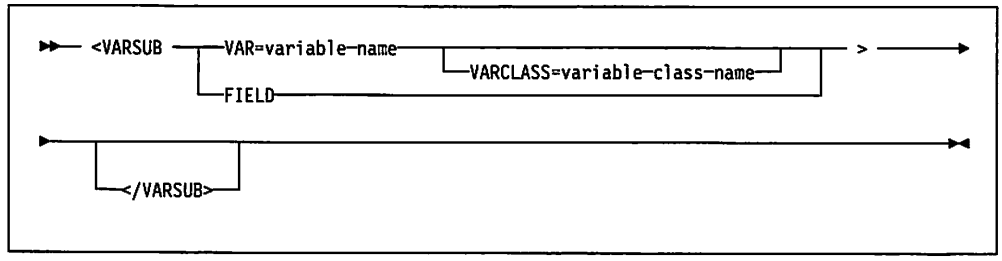
```
<!doctype dm system>

<varclass name=vflag Type='CHAR 1' >
<varclass name=vheight Type='CHAR 2'>
<varclass name=vweight Type='CHAR 3'>
<varlist>
   <vardcl name=frame varclass=vflag>
   <vardcl name=height varclass=vheight>
   <vardcl name=weight Varclass=vweight>
</varlist>

<panel name=stats>Health Assessment
   <area>
     <dtacol pmtwidth=15 entwidth=20 selwidth=20>
       <selfld type=single>What is your body frame?
         <choice checkvar=frame match=1>Large
         <choice checkvar=frame match=2>Medium
         <choice checkvar=frame match=3>Small
       </selfld>
       <dtafld datavar=height>Enter your height in inches
       <dtafld datavar=weight>Enter your weight
     </dtacol>
   </area>
</panel>
```

VARSUB (Variable Substitution)

The VARSUB tag defines variables substituted in a message definition.



VAR = variable-name

This names the variable whose value should be substituted at run time. The named variable must be declared using the VARDCL tag. The VARCLASS tag named on that VARDCL tag defines the display attributes of the variable. These can be overridden by specifying a VARCLASS on the VARSUB tag.

VARCLASS = variable-class-name

This attribute specifies the name of the variable class, defined using the VARCLASS tag. This variable class overrides the default variable class referenced by the VARDCL that declared the data variable. The VARCLASS specified must have the TYPE information that matches the TYPE information of the overridden VARCLASS.

If the VARCLASS specified does not have TYPE information that matches the type information of the VARCLASS, a warning message is issued.

FIELD

The FIELD attribute indicates that the value to be substituted at run time should be taken from the current field or the currently active panel. The field attribute cannot be used if the VAR attribute is specified.

Description

The VARSUB tag defines variables substituted in a message definition. You code the VARSUB tag in the *message-text* of a MSG tag. The value of the variable is inserted by the Dialog Manager at the position in the message text where the VARSUB tag is coded.

If the FIELD attribute specified on the VARSUB tag is displayed using the MSG parameter of the DISPLAY service call, DM inserts a null value.

The VAR attribute names a dialog variable whose value will be retrieved from the variable pools and inserted into the message text when the message is used by a dialog.

The FIELD attribute indicates that the value to be inserted into the message text will be retrieved from the field being validated by DM when the display of the message is requested.

At run time, the substitution for FIELD occurs only for the messages issued as a result of field validation. Messages with the FIELD attribute specified on the VARSUB tag are usually used for field validation messages. They can be specified on the VARCLASS, CHECKL and XLATL tags.

Conditions

You can use the VARSUB tag only within a MSG tag.

Nested Tags

None.

Example

The following markup is an example of the VARSUB tag. Using the FIELD attribute on the VARSUB tag, you can also specify that the value to be substituted into the message when it is displayed should be the value that was entered into a field. For example, when validation of the field fails, the message that appears would read as follows: The value you entered, "125", is incorrect.

```
<!doctype dm system>

<varclass name=agecls type='char 3'>
  <checkl msg=agea000>
    <checki type=range parm1=1 parm2=120>
  </checkl>
</varclass>

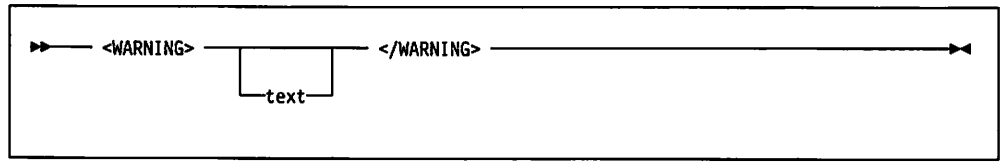
<varlist>
 <vardcl name=age varclass=agecls>
</varlist

<panel name=example1>VARSUB Example
 <dtafld datavar=age usage=in entwidth=3>AGE
</panel>

<msgmbr name=agea00>
 <msg suffix=0>The value you entered, "<VARSUB FIELD>", is incorrect.
</msgmbr>
```

WARNING (Warning)

The WARNING tag defines a statement that alerts the user of a possible risk.



text
This specifies the text of the warning.

Description

The WARNING tag defines a statement that alerts the user of a possible risk. Use a WARNING tag to alert users to a risk of possible error conditions in the system, or of a risk associated with a user interaction.

Code a warning statement before the text to which it pertains so that the user reads about the possible risks before reading the text. When a warning is displayed, the word "Warning: ", or its translated equivalent appears on the screen before the warning text.

Conditions

You can use the WARNING tag only within an information region. A WARNING tag must immediately follow (with no intervening blanks) a P, LI, or LP tag.

Nested Tags

You can code the following tags within the WARNING tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
FIG (figure)	Application Panel Help Panel	Multiple	15-70
LINES (lines)	Application Panel Help Panel	Multiple	15-95
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165
UL (unordered list)	Application Panel Help Panel	Multiple	15-176
XMP (example)	Application Panel Help Panel	Multiple	15-202

Example

The following markup of a help panel contains a warning statement. Figure 15-55 shows the formatted result.

```
<help name=warnin1>Help for Updating
  <area>
  <info>
    <p>Select the "Update" choice after specifying a file name.
    <p><warning>Updating a file changes the contents of the
    existing file. Any items deleted while in Update mode cannot be
    retrieved.
    </warning>
  </info>
  </area>
</help>
```

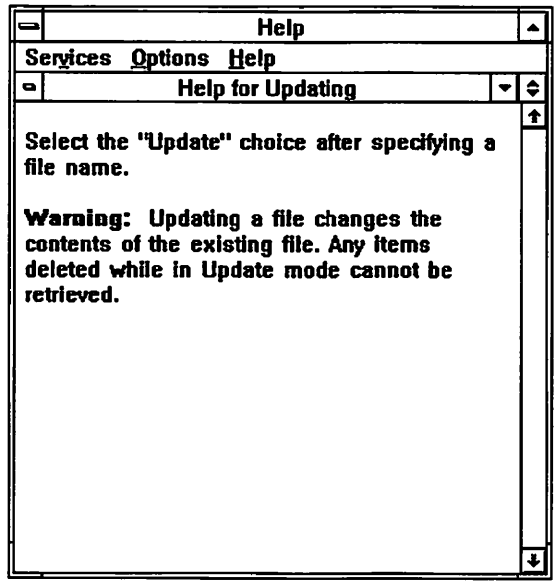
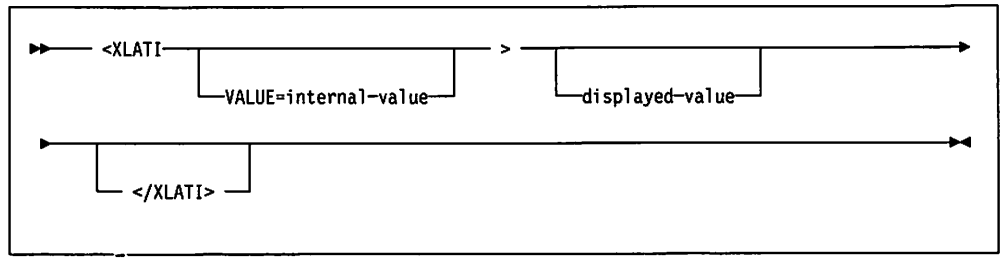


Figure 15-55. Warning Statement

XLATI (Translate Item)

The XLATI tag defines an individual list element in a translate list.



VALUE = internal-value

The Dialog Manager saves this value in the variable pool when translating on input and retrieves it from the variable pool when translating on output. If the *internal-value* contains characters other than A—Z and 0—9, you must enclose it in quotes.

When a match is found, the related *displayed-value* is used when this variable is displayed. If no match is found and there is not an XLATI tag coded without an internal value or displayed value, either a message is displayed or an error is returned to the application. For more information, refer to the section on user interaction in the *Dialog Manager Guide and Reference*.)

If there is an XLATI tag coded without an *internal value* and *displayed value*, and there is no explicit match, then processing continues. See "XLATL (Translate List)" on page 15-197 for more information. The specified internal value must adhere to special format when the enclosing VARCLASS type is ITIME, IDATE, STDTIME, STDDATE or NUMERIC. If the internal format does not conform to the proper format as outlined below, the compiler issues an error message. See "VARCLASS (Variable Class)" on page 15-178 for more information about these types:

Varclass Type	Internal Value Format	
ITIME	HHMM	(Hours, Minutes)
STDTIME	HHMMSS	(Hours, Minutes, Seconds)
IDATE	YYMMDD	(Year, Month, Day)
STDDATE	YYYYMMDD	(Year, Month, Day)
NUMERIC	[±] [..d] [p[d..]]	

The character *d* is a digit (0—9) and *p* is the decimal separator, which is always a period.

If this attribute is omitted, the Dialog Manager does not translate the *displayed-value* before storing it in the pool. When translating on output, if this attribute is omitted, the Dialog Manager translates to the *displayed-value* any value that is not already matched.

displayed-value

This attribute specifies the displayed value that must be matched when doing a translation on input and the result when doing a translation on output. Any characters, including embedded blanks, are allowed in the *displayed-value*. If the value has trailing blanks that you want preserved, or the value consists of only blanks, the value should be coded within the LIT tag. If the LIT tag is not used, all trailing (not leading) blanks will be stripped and any value with only blanks will indicate that no value was specified.

On input, when a match is found, the related *internal-value* is assigned to the panel variable and no further processing takes place. If no match is found and there is not an XLATI coded without an internal value and displayed value, either a message is displayed or (if Exit or Cancel was requested by the user) an error is returned to the application. For more information, refer to the section on user interaction in the *Dialog Manager Guide and Reference*.)

On output, when a match is found, the related *displayed-value* is assigned to the panel variable and no further processing takes place. If no match is found and there is not an XLATI coded without an internal value and displayed value, an error is returned to the application. For more information, refer to the section on user interaction in the *Dialog Manager Guide and Reference*.)

If there is an XLATI coded without an internal value and displayed value, any XLATI tags coded after this are ignored. See "XLATL (Translate List)" on page 15-197 for additional information.

When translating on output, if this value is omitted the *internal-value* is not translated before being displayed. When translating on input, if this attribute is omitted, it means that any value not already matched is to translate to the *internal-value*.

Description

The XLATI tag defines an individual list element in a translate list. As many XLATI tags as necessary to accomplish the desired translation can be included within the translate list.

Because any value is acceptable when an XLATI tag is coded without an internal value and displayed value, any XLATI tags coded after this are ignored.

Each XLATI tag provides information necessary to translate a *displayed-value* to an internal variable value (the VALUE attribute) and vice versa. Translation is done in the order given by the tags. Translation stops when a match is found.

Conditions

XLATI tags must be coded within an XLATL tag.

Nested Tags

You can code the following tag within the XLATI tag:

Tag	Dialog Element	Usage	Page
LIT (literal)	None	Multiple	15-97

Example

The following markup contains a translate list that performs input and output translation on values assigned to the winter months. It also shows how an UPPER translate list is used to change the input value to a common format prior to applying a validity check.

```
<!doctype dm system>

<varclass name=winterc type='char 1'>
   <xlatl>
       <xlati value=0><lit> </lit>
       <xlati value=1>December
       <xlati value=1>DEC
       <xlati value=2>January
       <xlati value=2>JAN
       <xlati value=3>February
       <xlati value=3>FEB
       <xlati value=4>March
       <xlati value=4>MAR
       <xlati>
   </xlatl>

<varclass name=costc type='numeric 6 2'>
  <checkl>
     <checki type=range Parm1=0 Parm2=9999.99>
  </checkl>

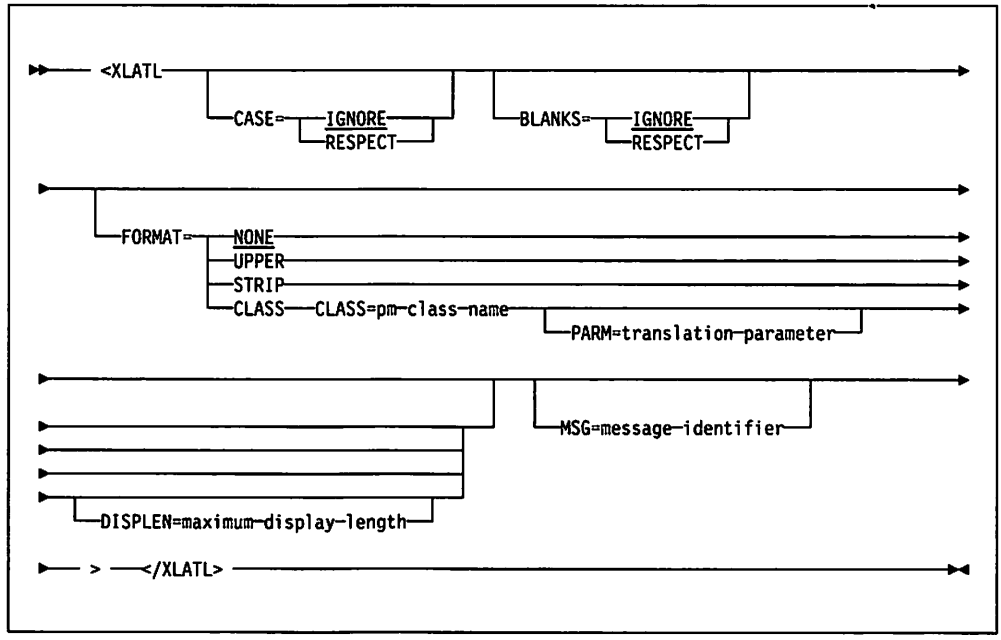
<varclass name=heatc type='char 5'>
   <xlatl format=upper>
   </xlatl>
   <checkl>
      <checki type=values Parm1=EQ Parm2= 'GAS OIL STEAM'>
   </checkl>

<varlist>
   <vardcl name=month varclass=winterc>
   <vardcl name=cost varclass=costc>
   <vardcl name=heat varclass=heatc>
</varlist>

<panel Name=xlati01>Heating Cost Survey
   <topinst>Complete the information below and then press Enter.
   <area>
     <dtacol Pmtwidth=15 Entwidth=30 Deswidth=25>
        <dtafld Datavar=month>Winter Month
          <dtafldd>Enter December-March or its abbreviation
        <dtafld Datavar=cost>Heating cost
        <dtafld Datavar=heat>Type of heating
     </dtacol>
   </area>
   <botinst>Thank you for your participation.
</panel>
```

XLATL (Translate List)

The XLATL tag identifies the beginning of a translate list for a variable class.



CASE = IGNORE | RESPECT

IGNORE means that the case of the letters a – z is ignored during comparisons, when trying to match the displayed value of a character variable on input, (or the internal value on output), with a list item. RESPECT means that case is to be taken into account.

CASE applies to single-byte characters in the context of the code page and country specified when the tag source is compiled.

This attribute applies only when FORMAT = NONE; otherwise, it is ignored.

BLANKS = IGNORE | RESPECT

IGNORE means that leading and trailing blanks are ignored during comparisons when trying to match the displayed or internal value of a character variable with a list element. RESPECT means that leading and trailing blanks are to be taken into account. If any of the XLATI tags specify display values with significant trailing blanks or a value of all blanks, then RESPECT should be specified on this attribute. This attribute operates on both single-byte character space and double-byte character space blanks.

This attribute, BLANKS, refers to the single-byte character space for VARCLASS type CHAR. It refers to both single-byte character space and double-byte character space for MIXED and ANY. It refers to the double-byte character space when all XLATI *displayed values* are double-byte character strings. For more information see "XLATI (Translate Item)" on page 15-194.

This attribute applies only when FORMAT = NONE; otherwise, it is ignored.

FORMAT = type of translation

This indicates the type of translation. The following attributes are the valid types:

NONE

NONE specifies that enclosed XLATI tags are to be used to translate the value on an item-for-item basis. NONE specifies that enclosed XLATI tags are to be used to translate the value on an item-for-item basis. This attribute is not supported for VARCLASS = DBCS.

UPPER

This specifies that the variable value should be translated to uppercase. UPPER applies to single-byte characters in the context of the code page and country code specified when the tag source is compiled.

STRIP

The STRIP attribute specifies that leading and trailing blanks should be removed from the value. This attribute operates on both single-byte character set and double-byte character set blanks.

This attribute, STRIP refers to the single-byte character set for VARCLASS type CHAR. It refers to both single-byte character set and double-byte character set for MIXED and ANY. It refers to the double-byte character set for VARCLASS type DBCS.

CLASS

The CLASS attribute specifies that a translate user exit performs the translation at run time. For more information, refer to the user exit section of the *Dialog Manager Guide and Reference*.

CLASS = pm-class-name

Indicates the name of the translate user exit to be used to perform a translation.

PARM = translation-parameter

The PARM attribute contains any specific parameters needed in the translation. Interpretation of this parameter is dependent on the type of translation being performed.

DISPLEN = maximum-display-length

This attribute specifies the maximum number of bytes needed to convert an internal value to its displayable format. The minimum value is 1 and the maximum is 32767. The default value is equal to the variable length specified or implied by the TYPE attribute on the enclosing VARCLASS tag.

MSG = message-identifier

This attribute specifies the ID of a message issued when the translation specified by the XLATL fails. An input translation will fail if a user enters a value not specified in the XLATI translate list. If there is no internal value, and no displayed value specified on the XLATI tag, any value not in the list is accepted without a message being returned. If no message ID is specified and an error occurs, a default message supplied by Dialog Manager is displayed.

Note: This message is not used if translation on output fails. If the value does not pass output translations, a return code is returned to the application and the panel is not displayed or is exited if currently displayed.

On input translations, (if the user specifies Exit or Cancel), a return code is returned to the application instead of displaying a message.

For more information, refer to the section on user interaction in the *Dialog Manager Guide and Reference*.

Description

The XLATL tag identifies the beginning of a translate list for a variable class. XLATI tags, which define the elements of the translate list, are coded within the XLATL tag. A translate list is defined within a VARCLASS tag.

If FORMAT=NONE is specified, Dialog Manager assumes that there are XLATI tags between the start and end XLATL. If something other than FORMAT=NONE is used and XLATI tags are coded within XLATL, a warning message is sent and the enclosed XLATI tags will be ignored. The CASE and BLANKS attributes are only valid if FORMAT=NONE.

Translate lists are optional and provide a means of translating between a displayed value and an internal value. Translation can occur on input (the value the user enters is translated before being put in the Dialog Manager variable pool), on output (the value from the Dialog Manager variable pool is translated before the user sees it), or both, depending on the usage specified on the referencing tag.

Translation on input occurs in the order the translations (XLATL tags) are coded. Translation on input implies validity checking, but non-matching values can be passed on for further validity checking by ommitting the internal value and display value on the XLATI tag. Because any value is acceptable when an XLATI tag is coded without an internal value and display value, any XLATI tags coded after this value are ignored.

Translate items are always searched from top to bottom. If an explicit match is found on an XLATI tag during inbound processing, any following translate lists and all validity checking lists are bypassed.

An explicit match is achieved during processing as follows:

On input, an explicit match occurs when the value the user enters matches one of the specified display values in the translate list. An explicit match also occurs if a display value is omitted (to indicate that any value is acceptable) and the corresponding internal value is specified.

On output, an explicit match occurs when the value from the variable pool matches one of the specified internal values in the translate list. An explicit match also occurs if an internal value is omitted (to indicate that any value is acceptable) and the corresponding display value is specified.

Omitting both the internal and display values does not produce an explicit match.

Note: Once an explicit match is found during translation processing, no other translate lists or items are searched and all validity checking is bypassed.

Translate lists are applied to variable values to be displayed or entered by the user. For variable values not displayed, the translate lists are ignored.

Conditions

XLATL tags must be coded before any CHECKL tags. The XLATL tag is coded within the VARCLASS tag and you can code Multiple XLATL tags within the VARCLASS tag.

Translation on input occurs only if the user modifies a value, and the translations apply to that value.

Nested Tags

You can code the following tag within the XLATL tag:

Tag	Dialog Element	Usage	Page
XLATI (translate item)	None	Multiple	15-194

Example

The following markup contains two translate lists that perform input and output translation on certain U.S. states and on products. These translate lists are used to convert an internal value to display value and back again.

```
<!doctype dm system>

<varclass Name=statec Type='CHAR 1'>
   <xlatl format=none>
       <xlati value=0><lit> </lit>
       <xlati value=1>North Carolina
       <xlati value=1>NC
       <xlati value=2>South Carolina
       <xlati value=2>SC
       <xlati value=3>Louisiana
       <xlati value=3>LA
       <xlati value=4>Florida
       <xlati value=4>FL
       <xlati value=5>Georgia
       <xlati value=5>GA
       <xlati>
   </xlatl>

<varclass name=productc type='char 1'>
   <xlatl format=none>
      <xlati value=0><lit> </lit>
      <xlati value=1>Nuts
      <xlati value=2>Bolts
      <xlati value=3>Screws
   </xlatl>

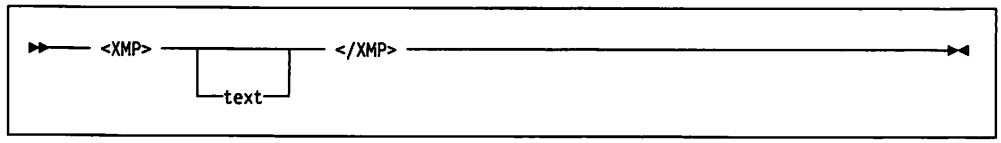
<varclass name=datec type=stddate>

<varlist>
   <vardcl name=state varclass=statec>
   <vardcl name=product varclass=productc>
   <vardcl name=sdate varclass=datec>
   <vardcl name=edate varclass=datec>
</varlist>

<panel name=xlatl01>Regional Sales Analysis
   <topinst>Complete the information below.
   <area>
     <dtacol pmtwidth=15 entwidth=30 deswidth=25>
        <dtafld datavar=state>State
          <dtafldd>Enter the name of the U.S. state or its abbreviation.
        <dtafld datavar=product>Product
        <dtafld datavar=sdate>Start date
        <dtafld datavar=edate>End date
     </dtacol>
   </area>
   <botinst>Press Enter to generate a sales report.
</panel>
```

XMP (Example)

The XMP tag defines an example that will be formatted, in monospace font, exactly as it is entered.



text
This specifies the unformatted text.

Description

The XMP tag defines an example that will be formatted, in monospace font, exactly as it is entered. The XMP text will appear indented two spaces from the current left margin.

The body of an example is composed of all the text between the XMP tag and its matching end tag.

Text between the XMP tag and its matching end tag will be treated the same as the text between the LINES tag and its matching end tag. No word-wrapping is done. Text too long to fit on a line is truncated. Tags that normally cause word-wrapping (for example, P, LI, and CAUTION) will not cause word-wrapping when nested within an XMP tag.

Note: The XMP tag uses a monospace font. When it is displayed, text coded with a monospace font will line-up exactly as it is coded.

Conditions

You can code an XMP tag within an information region.

You can code multiple XMP tags, as long as they are not nested anywhere within each other.

Nested Tags

You can code the following tags within the XMP tag:

Tag	Dialog Element	Usage	Page
DL (definition list)	Application Panel	Multiple	15-50
NOTE (note)	Application Panel Help Panel	Multiple	15-119
NT (note)	Application Panel Help Panel	Multiple	15-121
OL (ordered list)	Application Panel Help Panel	Multiple	15-123
P (paragraph)	Application Panel Help Panel	Multiple	15-125
PARML (parameter list)	Application Panel	Multiple	15-137
RP (reference phrase)	Help Panel	Multiple	15-152
SL (simple list)	Application Panel Help Panel	Multiple	15-165

Tag	Dialog Element	Usage	Page
UL (unordered list)	Application Panel Help Panel	Multiple	15-176

Example

In the following markup, an XMP tag is being used to show how an entry should be made. Figure 15-56 shows the formatted result.

```
<panel name=xmp1 width=55>Address Panel
  <area>
  <info>
    <p>Type your address as in the following example.
    <xmp>
Address:  8 Cherry Street
City:     Morrisville
State:    VT
Zip:      05672
    </xmp>
  </info>
  </area>
</panel>
```

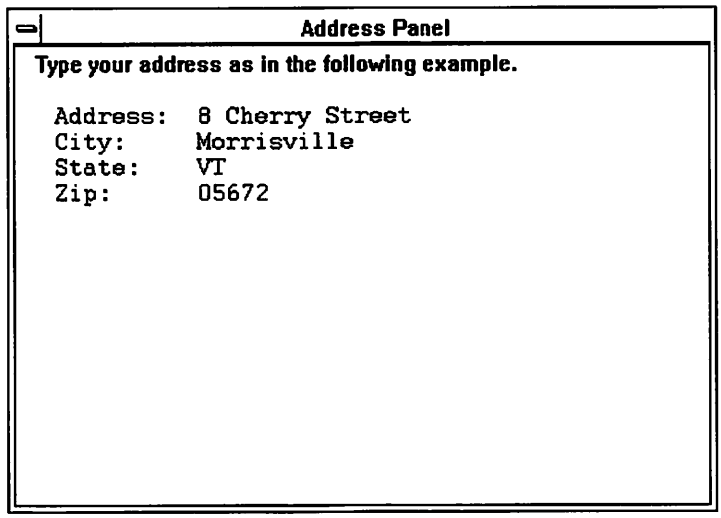
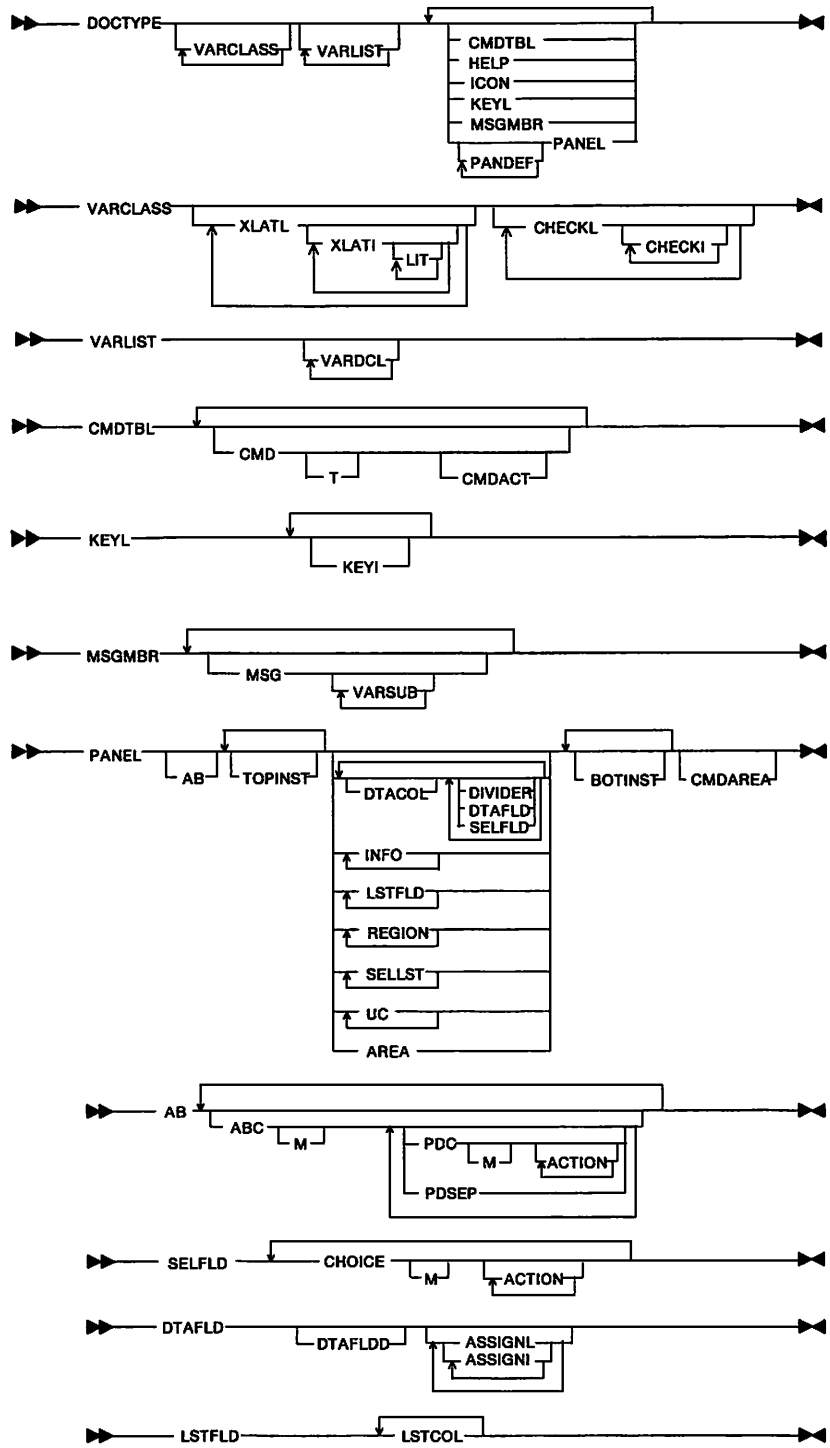
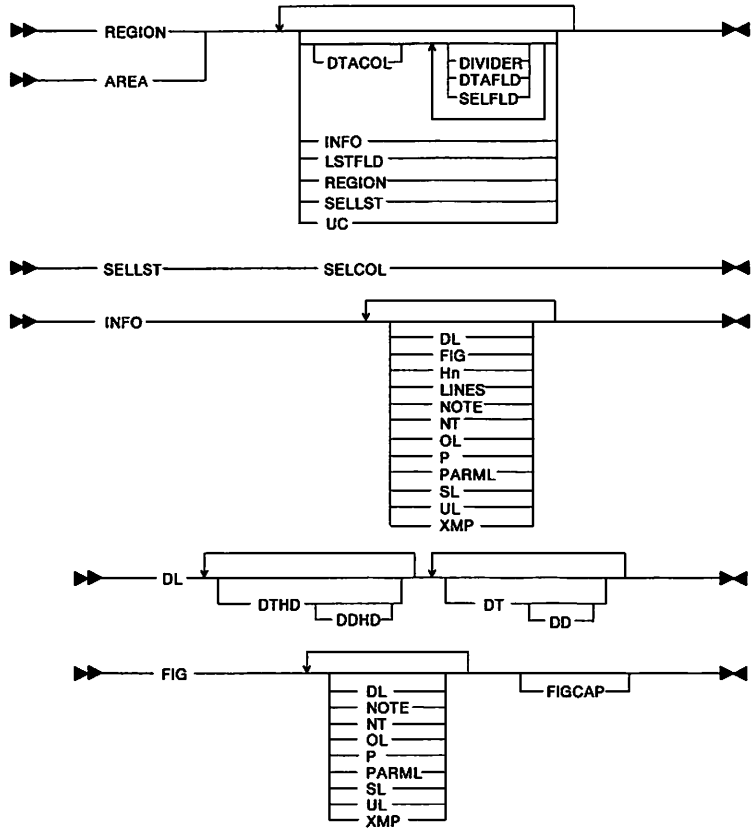


Figure 15-56. Example

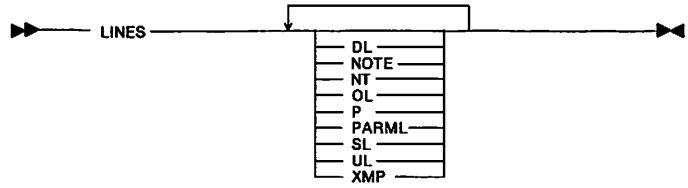
Appendix A. Nesting Order of Dialog Tag Language Tags

The following diagrams can be used as a guide for the nesting of DTL tags.

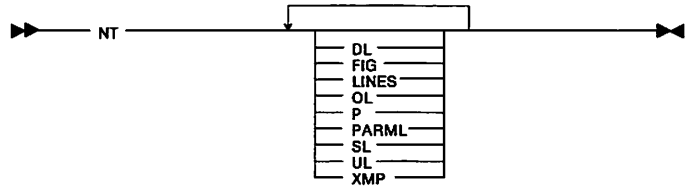




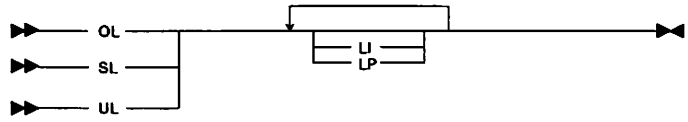
Note: You cannot nest FIG or LINES anywhere within the FIG tag.

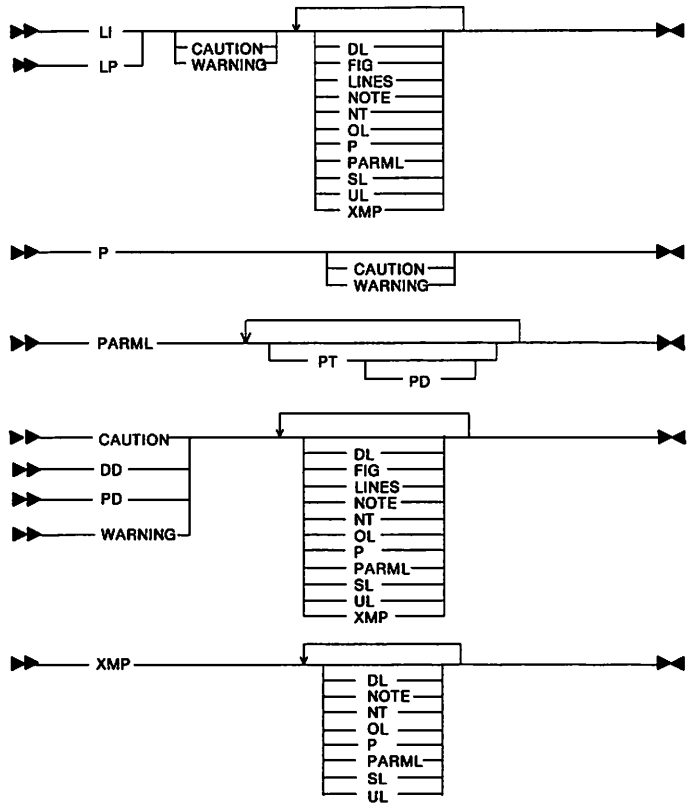


Note: You cannot nest FIG or LINES anywhere within the LINES tag.

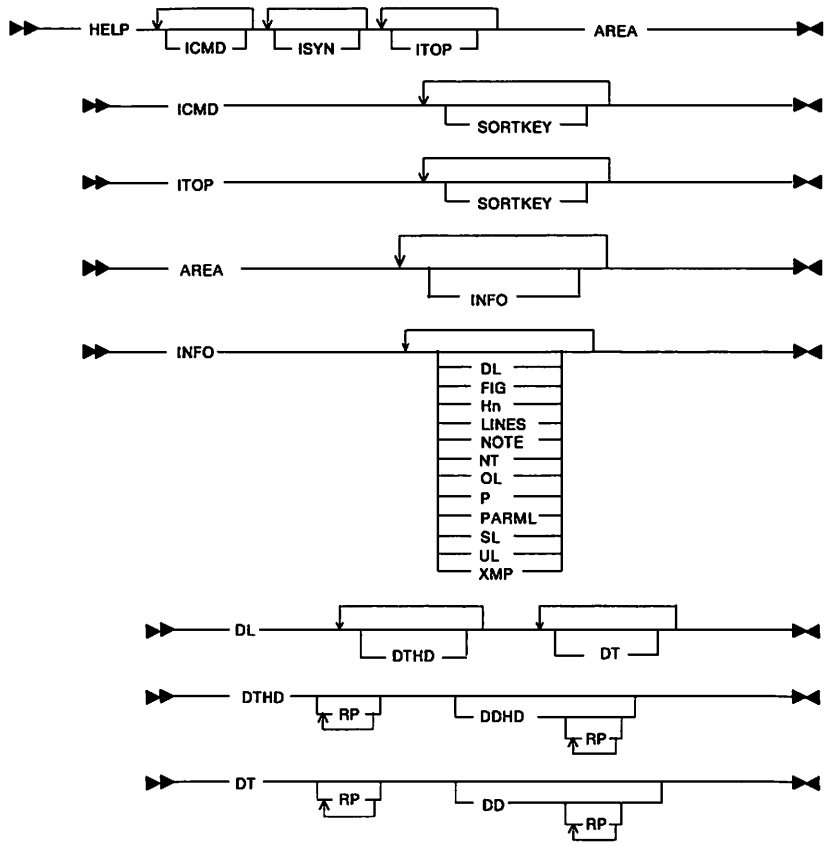


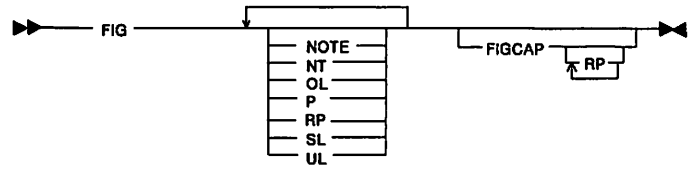
Note: You cannot nest NOTE or NT anywhere within the NT tag.



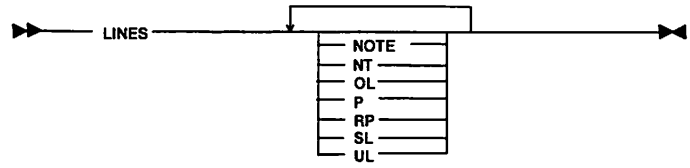


Note: You cannot nest XMP, LINES, or FIG anywhere within the XMP tag.

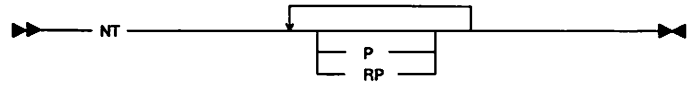




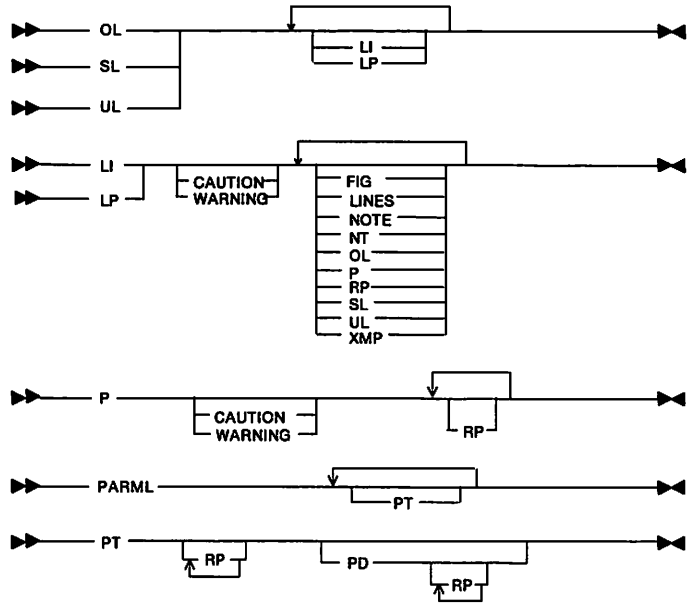
Note: You cannot nest FIG or LINES anywhere within the FIG tag.

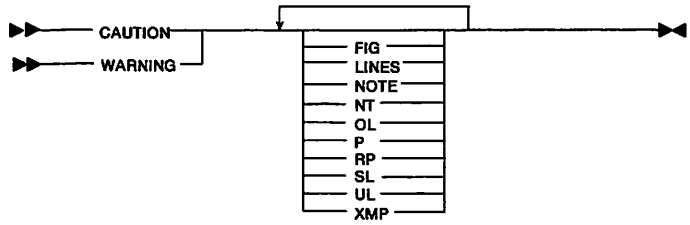


Note: You cannot nest FIG or LINES anywhere within the LINES tag.

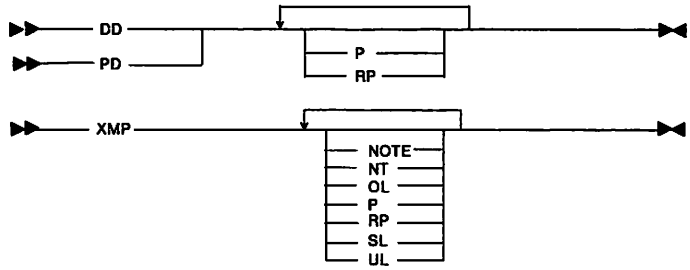


Note: You cannot nest NOTE or NT anywhere within the NT tag.





Note: You cannot nest CAUTION or WARNING anywhere within the CAUTION or WARNING tags.



Note: You cannot nest FIG, LINES, or XMP anywhere within the XMP tag.

Glossary

This glossary defines terms as they are used by the Dialog Manager. For additional definitions, refer to the *Dictionary of Computing*, SC20-1699.

A

action. One of the defined tasks that an application performs. Actions modify the properties of an object or manipulate the object in some way. Users may choose an action in several ways: by typing a command, pressing a function key, selecting the action from an action bar pull-down, or by selecting a pushbutton.

action bar. The area at the top of a primary window that contains keywords that give users access to actions available in that window. After users select a choice in the action bar, a pull-down extension appears from the action bar. Also known as *menu bar*.

action bar pull-down. An extension of the action bar that displays a list of actions related to a selected choice in the action bar. After users select a choice in the action bar, the pull-down appears. Additional pop-up windows can appear from pull-down choices. Also known as *menu*.

action message. Information that appears in a message pop-up. It tells users that an exception condition has occurred. Users must perform an action to correct the situation. Compare with *information message* and *warning message*.

application. A collection of software components that people use to perform specific types of work on a computer.

application ID. A 4-character identifier that you specify on the DMOPEN call. The *application-id* identifies the command table and application library definition file.

assignment list. A list that contains values assigned to one variable based on the content of another variable during panel processing.

attribute. A keyword, and its associated value, that specifies a particular characteristic (for example the name, size, or type) of items such as a field, panel, or command table.

available choice. An item that the current state of the application allows users to select.

C

cascade. To arrange windows and icons so that the top left corner of each is below and to the right of the one it overlaps. The windows appear like index cards stacked one behind the other.

character. A symbol used in printing and viewing. For example, a letter of the alphabet, a numeral, a punctuation mark, or any other symbol that represents information.

character unit. A unit of measure of horizontal and vertical space that is dependent on the system or display device. The average character unit is the width of a lowercase "o."

check box. A choice containing a square box with descriptive text beside it. It acts like a switch. An "X" appears in the check box to show that a choice is selected. Check boxes can be used alone or grouped in related sets so users can choose one or more choices. Check boxes are provided through Dialog Manager's multiple-choice selection field.

check item. A DTL tag that defines a single test of an input value. This test applies to the value the user enters for a data field or a list field.

check list. A list of check items.

client area. The part of a window inside the border that is below the action bar. It is the user's work space, where users type information and choose selections from selection fields. It can contain the following panel elements: top instructions, panel body, bottom instructions, and the command area.

command. The name and parameters associated with an action that can be performed by a program. Commands are associated with pull-downs, selection fields, function keys, and the command area.

command area. A panel element composed of a command prompt and a command entry field.

command area prefix. A displayed symbol that indicates where the user enters commands. The default is ===>.

command entry field. The part of the command area into which users type commands.

command field prompt. A word or words that identify a command entry field. The default is "Command" or its translated equivalent.

command procedure. A program written in Procedures Language 2/REXX.

contextual help. Specific information about the item that the cursor is on. This help information is *contextual* because it describes information about a specific item as it is currently used. The information provided is specific to the meaning of the item within the work session. Also known as *field-level help*. Contrast with *extended help*.

cursor. A visual cue that shows users the current position of the keyboard input focus. The keyboard cursors are the *selection cursor* and the *text cursor*.

D

data field. A field that can accept input or display output or both. A data field also includes the prompt text and description text. See also *entry field* and *output field*.

DBCS. See *double-byte character set*.

declaration subset. A part of the DOCTYPE declaration statement where entity and parameter entity declarations are made.

de-emphasize. To use shading, color, or half-toning to indicate that an item on a panel, such as a pull-down choice, is unavailable. Also known as *graying*.

dialog. The communication between a person and a computer. Within the Dialog Manager, a dialog starts when a DM application issues the DMOPEN service to establish communication with the Dialog Manager. All subsequent service calls, to display a panel or to define a variable, belong to the same dialog. A dialog ends when the DM application issues the DMCLOSE call or when the application ends.

dialog element. A part of a DM application defined with the Dialog Tag Language (DTL). Dialog elements include panels, message members, key mapping lists, icons, and an application command table. You define the dialog elements with DTL tags in a source file. Dialog elements are later compiled and stored on disk.

dialog interface. An SAA definition of services and other facilities that can be used in writing DM applications.

Dialog Manager. A set of services and facilities that complement those of the underlying operating system to provide a highly productive method of developing an application to carry on an interactive dialog with a user.

Dialog Tag Language (DTL). A markup language used to define DTL elements. DTL is based on the Standard Generalized Markup Language (SGML).

dialog variable. A variable that is used to communicate data between the DM application and Dialog Manager.

dialog variable pool. A collection of dialog variables associated with the DM communication area. Only a program having access to the DM communication area used at the time the pool was created can access the variables in the pool.

divider. A solid or blank line that provides users with a visual distinction between two adjacent portions of an application panel. Also known as *separator*.

DM application. An application comprised of programs, panels, and messages that use the services and facilities provided by the Dialog Manager.

DM communication area. An area of program storage that the Dialog Manager uses to communicate status to, and process information for, the DM application.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages, such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, entering, displaying, and printing DBCS characters requires hardware and supporting software that are DBCS-capable.

DTL. See *Dialog Tag Language*.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

entity. A collection of characters that can be referenced as a unit in a DTL markup source file. Entities may be contained within the source file they are referenced from or within external files.

entry field. The portion of a data field into which users can type. See also *data field*.

explicitly defined variable. A dialog variable that is defined when a program uses the VDEFINE service with that variable name.

extended help. Information about the contents of the application window from which users requested help. Also known as *general help*. Contrast with *contextual help*.

F

field. A visual panel element. Fields can display variable data or accept user input. The types of fields that Dialog Manager supports are: data fields, list fields, selection fields, and selection lists.

field prompt. Descriptive, static text that identifies a selection field or an entry field.

function key. A key that causes a specified sequence of operations to be performed when it is pressed. An example is F1 for Help.

function key area. The area at the bottom of a panel that contains the function key assignments in pushbuttons.

H

help. Information about a specific field, an application window, or a pop-up window.

help for help. Information provided to users about how to use help.

help index. An index, with search capability, of all the help information in the application.

help panel. A panel that contains information to assist users. It is displayed in a help window.

help window. A Common User Access-defined window that contains information to assist users.

I

icon. A pictorial representation of an object or a selection choice. Icons can represent objects that users want to work on or actions that users want to perform. A unique icon also represents the application when it is minimized.

implicitly defined variable. A variable that the Dialog Manager defines when a Dialog Manager service refers to a dialog variable name that is not found in the dialog variable pool, or when a Dialog Manager service must store data in a dialog variable that does not already exist in the dialog variable pool.

information message. Information that appears in a message pop-up. It tells users that a function is performing normally or has performed normally. Compare with *action message* and *warning message.*

Information Presentation Facility. An OS/2 programming tool that can be used to develop an SAA Common User Access-conforming help interface.

input field. See *data field.*

input focus. The area of a window where keyboard interaction is possible.

instance identifier. An identifier the Dialog Manager assigns to a dialog when the dialog starts. Dialog Manager uses the instance ID to manage the resources associated with the dialog.

K

keys help. A listing of an application's keys and their assigned functions.

L

list box. A field that contains scrollable choices from which users can select. Also known as *selection list.*

M

markup. Text that is added to document data in order to convey information about it. There are three types of markup that the DTL uses: tags, references, and markup declarations.

markup declaration. Markup that controls how other markup of a document is to be interpreted, for example document type and entity declarations.

markup language. A set of characters, conventions, and rules to control the interpretation of document data. The Dialog Tag Language is a markup language.

message. Information not requested by users but shown by the computer in response to a user action or an internal process. Messages appear in a message pop-up.

message member. A dialog element that contains messages grouped together for use by the application.

message pop-up. A type of window that displays messages to users. Also known as *message box.*

message type. Indicates the severity of different types of messages. A message can be one of three types: action, warning, or information. An action message has the highest severity, a warning message has the next-highest, and an information message has the lowest.

mixed data. Data that contains both single-byte (SBCS) and double-byte characters (DBCS).

mnemonic. A single character, within the text of a choice, identified by an underscore. When users type a choice's mnemonic, that choice is selected.

modal. A method of operation that requires the user to complete interaction with a pop-up window before continuing to work in the window from which the pop-up was displayed.

mouse. An instrument used to move the pointer on the screen.

N

name list. A list of dialog variable names passed as a single parameter in a dialog service call.

O

output field. A type of data field that displays variable data. Users cannot enter data into an output field. See also *data field*.

P

panel. A particular arrangement of user interface components, such as an action bar, entry fields, selection fields, and list fields presented to users in a window.

panel body. A panel element that serves as the main work area of the panel.

The panel body of an application panel can contain: data fields, selection fields, selection lists, list fields, and information regions. An application panel can contain a scrollable area if defined with an AREA tag.

The panel body of a help panel contains help text and is scrollable.

panel definition. A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format of information to be presented to users in a window.

panel element. A part or parts of a panel definition. A single panel element (such as an action bar) can contain other panel elements, such as action bar choices. Panel elements can also be non-visual, such as a region.

panel ID. A panel element located to the left of the window title that identifies that particular panel. If the panel ID is visible, it appears in the left margin of the title bar and is separated from the window title with a hyphen (-).

parameter. A keyword whose value specifies characteristics for the Dialog Manager service with which it is associated.

parameter entity. An entity that is referenced from a markup declaration.

pointer. The symbol displayed on the screen that is moved by a pointing device, such as a mouse. It is also used to point at the objects users want to select and at actions they choose to perform on the selected objects.

pointing device. An instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pop-up window. A movable window, fixed in size, that extends the user's dialog with another window. Therefore, pop-up windows are associated with other underlying windows and appear when the application wants to extend the dialog in the underlying window. Users must finish interacting with the pop-up window before continuing with the dialog in the related window.

primary window. The window in which the main dialog between users and the application takes place. In a multitasking environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward.

program variable. A named changeable value that can exist only within a program. Its value cannot be obtained or used when the program that contains it is no longer active.

pull-down. See *action bar pull-down*.

pushbutton. A rounded-corner rectangle with text inside. Pushbuttons are used in pop-up windows for actions that occur immediately when the pushbutton is selected. Pushbuttons are provided through Dialog Manager's immediate-action selection field.

R

radio button. A choice containing a circle with descriptive text beside it. Radio buttons are combined to show users a fixed set of choices that are mutually exclusive. These fields must contain at least two choices, one of which is always selected. The circle is partially filled when a choice is selected. Radio buttons are provided through Dialog Manager's single-choice selection field.

reason code. A field in the DM communication area. The reason code further defines the return code from a dialog service call.

reference phrase help. User-selectable words or phrases within a help window for which additional information is available.

return code. A numeric code indicating the result of an operation. The return code is in the first 4 bytes of the DM communication area.

S

SBCS. See *single-byte character set.*

screen. The physical surface of a display device upon which information is presented to users.

scroll bar. A window component associated with a scrollable area, that provides users a visual cue that more information is available. Users scroll the information in the window by interacting with the scroll bar.

selection field. A field that contains a set of selectable choices.

selection list. A field that contains scrollable choices from which users can select. Also known as *list box.*

shift-in (SI) character. In EBCDIC, a hexadecimal X'0F' character that indicates the end of a DBCS character string in mixed data.

shift-out (SO) character. In EBCDIC, a hexadecimal X'0E' character that indicates the beginning of a DBCS character string in mixed data.

single-byte character set (SBCS). A set of characters in which each character is represented by one byte.

stacked VDEFINE. Two or more VDEFINE calls without corresponding VDELETE calls using the same variable name.

system extensions. Services and facilities that provide system-dependent function. System extensions are not part of the Systems Application Architecture.

system variable. Variables provided by the dialog interface that communicate special information between the DM application and the Dialog Manager. System variable names always begin with the character Z.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its identification.

tutorial. An application-defined action that provides access from the current window to a tutorial, if the application has one.

U

unavailable choice. An item that the current state of the application does not allow users to select because of some condition in the application. Unavailable choices are displayed with unavailable emphasis. Contrast with *available choice.*

unavailable emphasis. A visual cue that shows users which items in a list of choices are currently unavailable for selection. See also *de-emphasize*

W

warning message. Information that appears in a message pop-up. It tells users that a potentially undesirable situation could occur. Users only need to respond to the message to continue. Corrective action may be required later to avoid an error situation.

window. An area of the screen with visible boundaries through which information is displayed. A window can be smaller than or equal in size to the screen. Windows can overlap on the screen and give the appearance of one window being on top of another.

window title. The area in the title bar that identifies the information in the panel.

Special Characters

% notation. A special notation used to distinguish a variable name from a specific value when both can be assigned to a tag attribute. Precede the variable name with the percent (%) sign. When the % sign is used, the attribute value must be enclosed in quotes.

Index

A

AB (action bar) tag
 See also action bar
 description of 15-3
 example of 3-8, 3-10, 15-3
 how to code 3-7
 syntax for 15-3
ABC (action bar choice) tag
 description of 15-5
 example of 3-8, 15-6
 syntax for 15-5
ABCHELP.GML 3-11
action
 See also ACTION tag
 See also immediate-action field
 definition of X-1
 for selection choice 15-29
ACTION attribute
 in CMDACT tag 15-37
action bar
 See also AB (action bar)tag
 AB (action bar) tag defining 15-3
 ABC (action bar choice) tag within 15-5
 defining 3-6
 defining mnemonics 3-9
 definition of X-1
 for application panel 1-4
 for help panels 1-6, 3-11
 M (mnemonic) tag within 15-110
 PDC (pull down choice) tag within 15-142
 PDSEP (pull-down separator) tag within 15-145
 providing help for 3-10
 tags to use 3-7
action bar choice
 See also ABC (action bar choice) tag
 text 15-5
action bar choice (ABC) tag
 description of 15-5
 example of 15-6
 syntax for 15-5
action bar pull-down
 See also pull-down choice
 defining current sort option 3-14
 definition of X-1
action bar (AB) tag
 description of 15-3
 example of 15-3
 syntax for 15-3
action message
 defining 8-3
 definition of X-1
 description of 8-3
ACTION tag
 defining immediate-action field 5-17
 description of 15-7
 example of 15-9
 for pull-down choice 3-8
 for selection field action 5-18
 syntax for 15-7
ADDPOP service
 positioning pop-up window 15-60, 15-82
alerting users 6-26
ALIAS command action
 defining 9-5
 description of 9-5
 in CMDACT tag 15-37
ALIAS parameters
 in CMDACT tag 15-38
ALIGN attribute 5-12
 in DTAFLD tag 15-62
 in LSTCOL tag 15-103
aligning variable data 5-12, 5-34
ALPHA attribute
 in CHECKI tag 15-22
alphabetic test
 description of 4-16
 example of 4-16
ANY attribute
 in VARCLASS tag 15-179
application
 See also DM application
 definition of X-1
application command table
 See also commands
 CMD (command definition) tag defining 15-34
 CMDACT (command action) tag within 15-37
 CMDTBL (command table) tag within 15-42
 defining commands for 9-1
 defining with CMDTBL tag 9-1
 description 1-7
 description of 15-42
 establishing variable access 15-188
 T (truncation) tag within 15-169
application ID
 definition of X-1
application identifier
 in CMDTBL tag 15-42
application panel
 See also panel
 action bar 1-4
 bottom instruction 1-4
 command area 1-5
 function key area 1-5
 layout 1-4
 overview of 1-4
 panel body 1-4

application panel *(continued)*
panel ID/window title 1-4
top instruction 1-4
application-id
ISP 9-1
ISPS 9-1
using with CMDTBL tag 9-1
APPLID attribute 9-1
in CMDTBL tag 15-42
area
AREA tag defining 15-11
area divider (DIVIDER) tag
description of 15-48
example of 15-48
syntax for 15-48
AREA tag
defining scrollable area 3-15
description of 15-11
example of 3-15, 15-12
syntax for 15-11
AREA (area) tag
example of 3-15
array variable
defining dimensions of 4-2
assigned value
in ASSIGNI tag 15-14
ASSIGNI (assignment list item) tag
description of 15-14
example of 15-15
syntax for 15-14
ASSIGNL (assignment list) tag
description of 15-16
example of 15-17
syntax for 15-16
assignment list
definition of X-1
assignment list item (ASSIGNI) tag
description of 15-14
example of 15-15
syntax for 15-14
assignment list (ASSIGNL) tag
description of 15-16
example of 15-17
syntax for 15-16
attributes and values 2-1
attribute, definition of X-1
attribute, tag
description of 2-1
auto clear attribute
in DTAFLD tag 15-62
AUTOCLR attribute 5-13, 5-34
in DTAFLD tag 15-62
available choice
definition of X-1
AVAILVAR attribute 3-12, 5-20
in CHOICE tag 15-28
in CMD tag 15-35
in PDC tag 15-142

B

BACKWARD attribute
in CMDACT tag 15-38
BLANKS attribute
in ASSIGNL tag 15-16
in XLATL tag 15-197
BOTINST (bottom instruction) tag
description of 15-18
example of 3-19, 15-18
syntax for 15-18
bottom instruction
defining 3-19
description of 1-4
bottom instruction (BOTINST) tag
syntax for 15-18
tag description 15-18
tag example 15-18
BREAK attribute
in DL tag 15-50
in PARML tag 15-137

C

CANCEL attribute
in CMDACT tag 15-38
caption
for figure 15-73
CASE attribute
in ASSIGNL tag 15-16
in XLATL tag 15-197
CAUTION tag
description of 6-30, 15-20
example of 6-30, 15-21
syntax for 15-20
CHAR attribute
in VARCLASS tag 15-178
character test
description of 4-16
example of 4-16
character unit, definition of X-1
character variables
description of 4-3
example of 4-3
types of 4-3
character, definition of X-1
CHARS attribute
in CHECKI tag 15-22
check box
definition of X-1
check item
definition of X-1
check list
definition of X-1
specifying message for 8-4
check mark
example of 3-14
for pull-down choices 3-14

check variable name
in CHOICE tag 15-29
in PDC tag 15-142
CHECKI (validity check item) tag
description of 15-22
example of 15-25
syntax for 15-22
checking values within a numeric range 4-15
CHECKL (validity check list) tag
description of 15-26
example of 15-27
syntax for 15-26
using MSG attribute 8-3
CHECKVAR attribute 3-14, 5-17
in CHOICE tag 15-29
in PDC tag 15-142
CHGDEFS attribute
in CMDACT tag 15-38
choice
See also selection field
action 15-29
defining for selection field 15-28, 15-156
defining mnemonic 5-19
defining space for 5-23
designating mnemonic for 15-110
multiple-choice 15-29
single-choice 15-29
text 15-5, 15-29
unavailable 5-20
choice description text
in CHOICE tag 15-29
choice name
in CHOICE tag 15-28
CHOICE tag
defining a selection field 5-13
CHOICE (selection choice) tag
description of 15-28
example of 15-31
syntax for 15-28
CLASS attribute
in ACTION tag 15-7
in CHECKI tag 15-23
in CMDACT tag 15-39
in UC tag 15-173
in VARLIST tag 15-188
in XLATL tag 15-198
using 4-1, 9-7
class name
in ACTION tag 15-7
in CHECKI tag 15-23
in CMDACT tag 15-39
client area
definition of X-1
CMD attribute
in KEYI tag 15-88
CMD (command definition) tag
assigning function key 10-1
description of 9-1, 15-34
CMD (command definition) tag *(continued)*
example of 15-36
syntax for 15-34
CMDACT (command action) tag
description of 9-1, 15-37
example of 15-39
syntax for 15-37
CMDAREA (command area) tag
description of 15-40
example of 3-25, 3-26, 3-27, 15-41
syntax for 15-40
using 3-25
CMDTBL (command table) tag
assigning function key 10-1
description of 9-1, 15-42
example of 15-43
syntax for 15-42
code page
with DTLC command 13-2
column
defining a scrollable list display area 15-108
defining widths 5-24, 5-32, 15-56
defining within list display area 15-101
column data
defining 5-24, 15-154
in LSTCOL 15-101
column name
in LSTCOL tag 15-101
column variable
in SELCOL tag 15-154
column widths, identifying 15-56
columns
defining 15-101
defining scrollable columns of data 15-163
COLWIDTH attribute
in LSTCOL tag 15-103
command
See also commands
defining command area 3-25
definition of X-1
command action
providing alternate 15-37
command action tag
using 9-4
command action (CMDACT) tag
description of 15-37
example of 15-39
syntax for 15-37
command area
CMDAREA (command area) tag defining 15-40
defining 3-25
defining cursor placement 3-5
definition of X-1
command area prefix
definition of X-1
command area (CMDAREA) tag
description of 15-40
example of 15-41

command area (CMDAREA) tag *(continued)*
syntax for 15-40
command definition (CMD) tag
description of 15-34
example of 15-36
syntax for 15-34
command entry field
definition of X-1
command field prompt
definition of X-1
command index tag
defining sort key 15-167
command procedure, definition of X-2
command prompt text
in CMDAREA tag 15-40
command table
See application command table
command table (CMDTBL) tag
description of 15-42
example of 15-43
syntax for 15-42
commands
See also command
application command table 9-1
associating with keys 15-88
declaring variables for 4-1
defining application command table 9-1
defining help for 15-79
defining sort key for 15-167
defining with CMD tag 15-34
defining with CMDACT tag 15-37
defining with CMDAREA tag 15-40
defining with CMDTBL tag 9-1, 15-42
description of application command table 1-7
DM-provided commands 9-3
DTLC command 13-2
DTLD command 13-10
Help Manager index 7-13
providing alternate 15-37
rules for DTL commands 12-1
specifying command action 9-4
specifying with the RUN attribute 3-9
to compile DTL elements 12-1
to display and test 12-1
to remove elements from library 12-1
to verify tag syntax 12-1
truncating 9-7, 15-169
comment delimiter 2-5
comments
examples 2-5
including in source files 2-5
compact
for ordered list 6-17
for simple list 6-12
for unordered list 6-14
COMPACT attribute
in DL tag 15-50
in OL tag 15-123
COMPACT attribute *(continued)*
in PARML tag 15-137
in SL tag 15-165
in UL tag 15-176
compiler message
description of 12-2
compressing a library 12-7
contextual help
definition of X-2
control text
in UC tag 15-174
control word 2-1
converting data
translate item tag 15-194
country-code
with DTLC command 13-2
creating a check 4-17
CSRINDEX attribute
in PANEL tag 15-134
CSRPOS attribute
in PANEL tag 15-134
current state indicator
for pull-down choice 3-14
cursor
defining placement of 3-4
definition of X-2
placing in data field 15-60
CURSOR attribute
in PANEL tag 15-134
cursor field
in PANEL tag 15-134
cursor placement
defining 3-4
example of 3-5

D

data
defining columns of 5-24, 15-101
defining scrollable columns of 15-163
data column
defining 5-24
example of 5-24
data column tag
description of 5-24
data column (DTACOL) tag
description of 15-56
example of 15-58
syntax for 15-56
data columns
DTACOL (data column) tag defining 15-56
data field
adding additional text 15-66
attributes 5-12
defining a field prompt 5-1
defining alignment of data 5-12
defining an associated message 5-12
defining column widths 15-56

data field *(continued)*
defining cursor placement 3-4
defining data columns 5-24
defining help for 5-11
defining input/output 5-5
defining multiple 5-7
defining width 5-7
definition of X-2
examples of 5-6
providing descriptive text 5-10
tailoring 5-12
data field description tag
description of 5-10
example of 5-10
data field description (DTAFLDD) tag
description of 15-66
example of 15-66
syntax for 15-66
data field (DTAFLD) tag
description of 15-60
example of 15-64
syntax for 15-60
data fields
DTAFLD (data field) tag defining 15-60
DTAFLDD (data field description) tag within 15-66
DATAVAR attribute 4-2
in DTAFLD tag 15-61
in LSTCOL 15-101
in SELCOL tag 15-154
date
See also time and date variables
display length 4-18
example of variables 4-6
in VARCLASS tag 15-180, 15-181
retrieving with ZIDATE 4-6
retrieving with ZSTDDATE 4-6
using IDATE 4-5
using STDDATE 4-5
DBCS
See double-byte character set
DBCS attribute
in VARCLASS tag 15-178
DBCS restrictions
for leading and trailing blanks 2-3
DD (definition description) tag
description of 6-18, 15-44
example of 6-19, 15-45
syntax for 15-44
DDHD (definition description header) tag
description of 6-18, 15-46
example of 6-21, 15-47
de-emphasize
definition of X-2
pull-down choice 3-12
selection field choice 3-9
declaration subset
definition of X-2
declaring document type 2-4
declaring variables 4-1
with VARDCL (variable declaration) tag 15-186
default key assignments 10-2
defaults
defining for panels 15-129
defining a selection list 5-26
defining command action 15-37
defining commands 15-34
defining messages 15-114
definition description
in DD tag 15-44
definition description header
in DDHD tag 15-46
definition description header (DDHD) tag
description of 15-46
example of 15-47
definition description (DD) tag
description of 15-44
example of 15-45
syntax for 15-44
definition list
description of 6-18
example of 6-19
definition list tag
description of 6-18
definition list (DL) tag
description of 15-50
example of 15-51
syntax for 15-50
definition term
in DT tag 15-54
definition term header
in DTHD tag 15-68
definition term header (DTHD) tag
description of 15-68
example of 15-68
syntax for 15-68
definition term (DT) tag
description of 15-54
example of 15-54
syntax for 15-54
deleting dialog elements 12-6, 13-10
delimiter symbol 2-1
depth 3-2
depth and width
defining with PANDEF tag 11-1
DEPTH attribute
in PANDEF tag 15-129
in PANEL tag 15-133
description text
in DTAFLDD tag 15-66
destination variable name
in ASSIGNL tag 15-16
DESTVAR attribute
in ASSIGNL tag 15-16
DESWIDTH attribute 5-24
in DTACOL tag 15-56

DESWIDTH attribute *(continued)*
in DTAFLD tag 15-62
dialog
definition of X-2
dialog element
creating 2-1, 12-2, 13-2
definition of X-2
deleting 12-6
for help panel 1-6
removing 13-10
dialog elements
defining 1-3
description of 1-1, 1-3
description of application command table 1-7
description of application panel 1-4
description of help panel 1-5
description of key mapping list 1-8
description of messages 1-7
dialog interface
definition of X-2
Dialog Manager
definition of X-2
Dialog Tag Language
advantages 1-1
coding attributes and values 2-1
coding tag text 2-2
comments 2-5
document type declaration 2-4
nesting tags 2-3
similarity to BookMaster 1-2
source files 2-1
syntax conventions 2-1
using 2-1
using the PANEL tag 3-1
Dialog Tag Language Guide and Reference
audience iii
organization iii
related publications iv
Dialog Tag Language (DTL)
See also tag
definition of X-2
introduction to 1-1
reducing tag usage 11-3
shortcuts 11-1
verifying syntax 12-2, 13-2
dialog variable
See also variables
definition of X-2
dialog variable pool
definition of X-2
DIR attribute
in REGION tag 15-149
in SELFLD tag 15-157
direction
specifying in regions 15-149
specifying in selection fields 15-157
DISPLAY attribute 5-13
DISPLAY service
positioning message pop-up 15-60, 15-82
using the panel name for display 3-1
display utility
restrictions 12-9
using 12-7
displayed value
in XLATI tag 15-194
displayed value attribute
specifying trailing blanks in 15-97
displaying panels and messages 12-8
displaying the current date 4-4
displaying the current time 4-4
DISPLEN
in XLATL tag 15-198
divider
definition of X-2
DIVIDER (area divider) tag
description of 15-48
example of 3-20, 3-24, 15-48
how to use 3-20
syntax for 15-48
DL (definition list) tag
description of 6-18, 15-50
example of 6-19, 15-51
syntax for 15-50
DM application
definition of X-2
providing help for 7-1
DM communication area
definition of X-2
DOCTYPE statement 2-4
description 14-2
parameters 14-2
syntax 14-2
document type declaration 2-4
document-type declaration 14-2
double-byte character set (DBCS)
definition of X-2
drive and path
specifying 12-7
DT (definition term) tag
description of 6-18, 15-54
example of 6-19, 15-54
syntax for 15-54
DTACOL (data column) tag
description of 15-56
example of 15-58
syntax for 15-56
DTAFLD (data field) tag
See also data field
description of 15-60
example of 15-64
syntax for 15-60
using 5-5
using MSG attribute 8-3
DTAFLDD (data field description) tag
See also data field

DTAFLDD (data field description) tag *(continued)*
description 15-66
example of 15-66
syntax for 15-66
DTHD (definition term header) tag
description of 6-18, 15-68
example of 6-21, 15-68
syntax for 15-68
DTL
See Dialog Tag Language
DTL commands
description of 12-1
rules for 12-1
DTL display utility
restrictions 12-9
using 12-7
DTLC
messages 12-2
DTLC command
description 13-5
description of 12-1
example 13-6
exit values 13-6
option values 13-7
output 13-6
parameters and options 13-2
rules for 12-1
syntax 13-2
using 12-2
DTLD command
description 13-11
description of 12-1
example 13-11
exit values 13-11
output 13-11
parameters and options 13-10
rules for 12-1
syntax 13-10
using 12-6

E

element
See dialog element
empty end tags 11-5
empty start tags 11-4
end tag delimiters 2-1
ENTER attribute
in CMDACT tag 15-38
entities
predefined 2-12
entity
defining 2-6
definition of X-2
description of 2-7
example 2-7, 2-8, 2-9, 2-10
naming conventions 2-7
parameter 2-10
entity *(continued)*
predefined 2-12
entity declarations
conditions 14-4
description 14-4
description of 14-3
example of 14-4
entry field
definition of X-2
ENTWIDTH attribute 5-7, 5-24
in DTACOL tag 15-56
in DTAFLD tag 15-62
example tag
description of 6-6
example of 6-7
example (XMP) tag
description of 15-202
example of 15-203
syntax for 15-202
EXHELP attribute
in CMDACT tag 15-38
EXIT attribute
in CMDACT tag 15-38
explicitly defined variable
definition of X-2
extended help
definition of X-2
external command name
in CMD tag 15-35
external command string
in ICMD tag 15-79

F

field
arranging on panel 15-149
defining a data field message 5-12
defining a list field message 5-34
defining data field 5-5
defining data field columns 5-24
defining immediate-action selection field 5-17
defining list field 5-30
defining multiple-choice fields 5-15
defining selection fields 5-13
defining single-choice fields 5-14
definition of X-3
providing additional information 6-31
FIELD attribute
in VARSUB tag 15-190
field data
in DTAFLD tag 15-61
field name
in DTAFLD tag 15-60
in SELFLD 15-156
field prompt
attributes 5-1
defining 5-1
definition of X-3

field prompt *(continued)*
example of 5-2
specifying width 5-4
field prompt text
in SELFLD tag 15-157
fields
defining 5-1
types of 5-1
FIG (figure) tag
description of 6-7, 15-70
example of 6-8, 15-72
syntax for 15-70
FIGCAP (figure caption) tag
description of 15-73
example of 15-73
syntax for 15-73
figure caption text
in FIGCAP tag 15-73
figure caption (FIGCAP) tag
description of 15-73
example of 15-73
syntax for 15-73
figure content
in FIG tag 15-70
figure tag
description of 6-7
example of 6-8
figure (FIG) tag
description of 15-70
example of 15-72
syntax for 15-70
fixed text 6-1
FKA
in CMDACT tag 15-38
FKA attribute
in CMDACT tag 15-38
in KEYI tag 15-89
FKA command
using 10-3
FORMAT attribute
in XLATL tag 15-198
FORWARD attribute
in CMDACT tag 15-38
fractional digits
in VARCLASS tag 15-182
FRAME attribute
in FIG tag 15-70
RULE value 2-2
function key
definition of X-3
KEYI (key item) tag defining 15-88
function key area
defining 10-1
definition of X-3
function keys
See keys

G

GAP attribute
in DIVIDER tag 15-48
GUTTER attribute
in DIVIDER tag 15-48

H

heading
example of 6-4
for description column of definition list 15-46
for term column of definition list 15-68
in the information region 6-4
levels of 6-4
heading tag
description of 6-4
example of 6-4
heading text
in Hn tag 15-77
heading (Hn) tag
description of 15-77
example of 15-78
syntax for 15-77
help
contents of ABCHELP.GML file 3-11
defining for a data field 5-11
defining help panels 7-1
defining help pull-down 3-11
defining index help 7-10
defining with HELP tag 15-75
definition of X-3
for action bar 3-10
for list field 5-33
for selection choice 5-19, 15-28
for selection field 5-19
for selection list 5-30
Help Manager index commands 7-13
in CMDACT tag 15-38
keys help 7-15
linking help to application 7-14
tags that contain the help attribute 7-14
tutorial 7-2
HELP attribute
in ABC tag 15-5
in CHOICE tag 15-28
in CMDACT tag 15-38
in DTAFLD tag 15-60
in HELP tag 15-75
in LSTCOL tag 15-102
in LSTFLD tag 15-106
in MSG tag 15-114
in PANDEF tag 15-129
in PANEL tag 15-132
in PDC tag 15-142
in RP tag 15-152
in SELFLD tag 15-156
in SELLST tag 15-161

HELP attribute *(continued)*
in UC tag 15-173
help for help
definition of X-3
help index
See also dialog element
defining synonyms 15-84
defining topics 7-10
definition of X-3
description of 1-6
help panel
See also dialog element
action bar 1-6
defining areas and regions 7-2
defining for a data field 5-11
defining help panel text 7-3
defining information region 15-82
defining reference phrases 7-8
defining with HELP tag 15-75
definition of X-3
description of 1-5
help text window 1-6
layout 1-5
main help window title 1-6
panel ID/help text window title 1-6
types of help 1-5
using the HELP attribute 3-4
using the INFO tag with 7-2
help panel name
in ABC tag 15-5
in CHOICE tag 15-28
in DTAFLD tag 15-60
in HELP tag 15-75
in LSTCOL tag 15-102
in LSTFLD tag 15-106
in MSG tag 15-114
in PANDEF tag 15-129
in PANEL tag 15-132
in PDC tag 15-142
in RP tag 15-152
in SELFLD tag 15-156
in SELLST tag 15-161
in UC tag 15-173
help panel tag
using 7-1
help panel title
in HELP tag 15-75
help panel (HELP) tag
example of 15-76
help panels
defining 7-1
help pull-down
defining 3-11
HELP tag
description of 7-1
help text window 1-6
help text window title 1-6
help topic
defining name of 15-86
defining synonyms 15-84
help window
definition of X-3
HELP (help panel) tag
description of 15-75
example of 15-76
HELPHELP attribute
in CMDACT tag 15-38
help, panel
description of 1-5
Hn (heading) tag
description of 6-4, 15-77
example of 6-4, 15-78
syntax for 15-77
hypertext 7-8

I

ICMD (index command) tag
defining sort key for 15-167
defining SORTKEY within 15-167
description of 7-13, 15-79
example of 15-79
syntax for 15-79
icon
See dialog element
icon name
in ICON tag 15-81
icon resource
creating an icon resource 15-81
icon resource (ICON) tag
description of 15-81
example of 15-81
syntax for 15-81
ICON (icon resource) tag
description of 15-81
example of 15-81
syntax for 15-81
ID attribute
in PANDEF tag 15-129
IDATE attribute
in VARCLASS tag 15-180
immediate-action
for pull-down choice 3-8
immediate-action field
defining 5-17
defining a default action 5-17
description of 5-17
implicitly defined variable
definition of X-3
implied paragraph
in LP tag 15-99
in DTAFLD tag
INDEX attribute
in CMDACT tag 15-38

index command tag
description of 7-13
index command (ICMD) tag
defining SORTKEY within 15-167
description of 15-79
example of 15-79
syntax for 15-79
index help
command 7-13
defining topics 7-10
using the ISYN tag 7-10
using the ITOP tag 7-10
index synonym (ISYN) tag
description of 15-84
example of 15-85
syntax for 15-84
index topic tag
defining sort key 15-167
index topic text
in ITOP tag 15-86
index topic (ITOP) tag
defining SORTKEY within 15-167
description of 15-86
example of 15-87
syntax for 15-86
index value
in PANEL tag 15-134
INFO (information) tag
description of 15-82
example of 15-83
syntax for 15-82
information message
defining 8-2
definition of X-3
description of 8-2
providing for fields 6-31
information region
defining 6-1
defining text 6-2
defining with INFO tag 15-82
description of 6-1
horizontal scroll bar 6-1
providing information for fields 6-31
tags for text 6-1
information regions
INFO (information) tag defining 15-82
information tag
description of 6-1
information (INFO) tag
description of 15-82
example of 15-83
syntax for 15-82
input
translating and saving 15-194
input area
defining with data field tag 15-60
input focus
definition of X-3
input/output data field
defining 5-5
instance identifier
definition of X-3
instruction text
BOTINST (bottom instruction) tag defining 15-18
example of 3-19
in BOTINST tag 15-18
in TOPINST tag 15-171
interactive fields 5-1
internal command name
in ACTION tag 15-7
in CMD tag 15-34
in KEYI tag 15-88
internal name
in XLATI tag 15-194
ISP application identifier 9-1
ISPS application identifier 9-1
ISYN (index synonym) tag
description of 15-84
example of 15-85
syntax for 15-84
item identifiers
in OL tag 15-123
in SL tag 15-165
in UL tag 15-176
item text
in LI tag 15-92
item translate list
description of 4-9
example of 4-11
ITIME attribute
in VARCLASS tag 15-179
ITOP (index topic) tag
defining sort key 15-167
defining SORTKEY within 15-167
description of 15-86
example of 15-87
syntax for 15-86

K

KEY attribute
in KEYI tag 15-88
key item
defining 15-88
key item (KEYI) tag
description of 15-88
example of 15-90
syntax for 15-88
key list name
in KEYL tag 15-91
in PANDEF tag 15-129
in PANEL tag 15-134
key mapping list
See also dialog element
defining 3-3, 10-1
description 1-8

key mapping list *(continued)*
description of 3-3
KEYI (key item) tag within 15-88
KEYL (key list) tag defining 15-91
using PANDEF tag 11-1
key mapping list (KEYL) tag
example of 15-91
key mapping (KEYL) list
defining 15-91
syntax for 15-91
KEYI (key item) tag
description of 10-1, 15-88
example of 15-90
syntax for 15-88
using 10-1
KEYL (key mapping list) tag
description of 10-1, 15-91
example of 15-91
syntax for 15-91
using 10-1
KEYLIST attribute
in PANDEF tag 15-129
in PANEL tag 15-134
keys
See also function key area
assigning actions 10-1
associating with commands 15-88
default key assignments 10-2
defining 10-3
defining a description of 15-88
defining a key mapping list 15-91
defining a sortkey 15-167
defining key mapping lists 10-1
defining keys for tasks 15-91
displaying 10-3
providing help for 7-15
using KEYLIST attribute of PANEL tag 3-3
KEYS attribute
in CMDACT tag 15-38
keys help
definition of X-3

L

LI (list item) tag
description of 15-92
example of 15-94
syntax for 15-92
library
compressing 12-7
defining a message member 15-116
deleting element from 12-6, 13-10
replacing and removing elements 12-4
specifying 12-3
LINES tag
description of 6-5, 15-95
example of 6-6, 15-96
syntax for 15-95
list
defining a list item 15-92
defining selection 5-26
definition 6-18
inserting text with list part tag 6-24
nesting lists within lists 6-25
ordered 6-15
ordered list (OL) tag 15-123
parameter 6-22
simple 6-11
simple list (SL) tag 15-165
types of 6-11
unordered 6-13
unordered list (UL) tag 15-176
list box
See selection list
list column
defining width 5-32
truncating 5-32
list column (LSTCOL) tag
description of 15-101
example of 15-104
syntax for 15-101
list columns
LSTCOL (list column) tag defining 15-101
list field
additional attributes for 5-34
auto-clear attribute 5-34
defining 5-30
defining alignment of data 5-34
defining an associated message 5-34
defining cursor placement 3-4
defining number of rows 5-33
defining required input for 5-34
description of 5-30
determining rows modified by user 5-33
drawing lines within 5-34
example of 5-31, 5-35
help for 5-33
sizing 5-36
tailoring 5-34
list field tag
example of 5-35
list field (LSTFLD) tag
description of 15-106
example of 15-108
syntax for 15-106
list fields
LSTCOL (list column) tag within 15-101
LSTFLD (list field) tag defining 15-106
list item (LI) tag
description of 15-92
example of 15-94
syntax for 15-92
list name
in LSTFLD tag 15-106
list part tag
description of 6-24

list part tag *(continued)*
example of 6-24
list part (LP) tag
description of 15-99
example of 15-100
syntax for 15-99
LIT (literal) tag
description of 15-97
example of 15-98
syntax for 15-97
literal display value
in LIT tag 15-97
literal (LIT) tag
description of 15-97
example of 15-98
syntax for 15-97
local variable classes 4-2
LP (list part) tag
description of 6-24, 15-99
example of 6-24, 15-100
syntax for 15-99
LSTCOL (list column) tag
description of 5-30, 15-101
example of 15-104
syntax for 15-101
using MSG attribute 8-3
LSTFLD (list field) tag
See also list field
description of 5-30, 15-106
example of 15-108
syntax for 15-106

M

M (mnemonic) tag
See also mnemonic
description of 15-110
example of 3-9, 5-19, 15-111
syntax for 15-110
using 3-9
main help window title 1-6
margin attributes
description of 3-16
example of 3-17, 3-18
MARGIND attribute 3-16
in AREA tag 15-11
MARGINW attribute 3-16
in AREA tag 15-11
markup
See also Dialog Tag Language (DTL)
definition of X-3
verifying syntax 12-2, 13-2
markup declaration 2-4
definition of X-3
markup declarations
comments 2-5
document type declaration 2-4
entity declarations 2-6
markup language
definition of X-3
description of 1-1
markup, coding 2-1
MATCH attribute 3-14, 5-17
in CHOICE tag 15-29
in PDC tag 15-143
MAXDEPTH attribute
in UC tag 15-174
MAXDIM attribute 4-2
declaring in list fields 15-101
in VARDCL tag 15-186
maximum display length
in XLATL tag 15-198
MAXROW attribute 5-29
MAXROWS attribute
in LSTFLD tag 15-107
in SELLST tag 15-162
MAXWIDTH 5-29
MAXWIDTH attribute
in LSTFLD tag 15-108
in SELLST tag 15-162
in UC tag 15-174
message
See also dialog element
See also message pop-up
compiler 12-2
declaring variables for 4-1
defining 3-3, 8-1
defining for a data field 5-12
defining for a list field 5-34
definition of X-3
description of 8-1
displaying 12-8
example of 1-7
for check list 8-3, 8-4
for data field 8-3
for failing specified translation 8-3
for failing validity check 8-3
for list column 8-3
member 8-1
MSG (message) tag defining 15-114
specifying a variable in text 8-4
specifying type of 3-3, 8-2
tags to define 8-1
types of 8-1
VARSUB (variable substitution) tag within 15-190
message id
in LSTCOL tag 15-102
message identifier
in CHECKL tag 15-26
in DTAFLD tag 15-61
in VARCLASS tag 15-183
in XLATL tag 15-198
specifying for an input field 15-61
message member
definition of X-3
establishing variable access 15-188

message member *(continued)*
MSGMBR (message member) tag defining 15-116
message member name
in MSGMBR tag 15-116
message member tag
description of 8-1
message member (MSGMBR) tag
description of 15-116
example of 15-116
syntax for 15-116
message pop-up
See also message
definition of X-3
positioning relative to data field 15-60
positioning relative to information region 15-82
message suffix number
in MSG tag 15-114
message tag
description of 8-1
message text
in MSG tag 15-114
message type
definition of X-3
message (MSG) tag
description of 15-114
example of 15-115
syntax for 15-114
messages
description 1-7
MINDEPTH attribute
in UC tag 15-174
minimization
empty end tags 11-5
empty start tags 11-4
null end tags 11-6
omitted optional end tags 11-4
unclosed start tags 11-6
MINROW attribute 5-29
MINROWS attribute
in LSTFLD tag 15-107
in SELLST tag 15-162
MINWIDTH attribute 5-29
in LSTFLD tag 15-108
in SELLST tag 15-162
in UC tag 15-174
MIXED attribute
in VARCLASS tag 15-178
mixed data
definition of X-3
mnemonic
See also M (mnemonic) tag
defining 3-7, 3-9
definition of X-3
for selection field choice 5-19
mnemonic character
in M tag 15-110
mnemonic (M) tag
description of 15-110
mnemonic (M) tag *(continued)*
example of 15-111
syntax for 15-110
MODVAR attribute
in LSTFLD tag 15-106
monospaced fonts
in LINES tag 15-95
mouse, definition of X-4
MSG attribute 5-12
in CHECKL tag 15-26
in DTAFLD tag 15-61
in LSTCOL tag 15-102
in VARCLASS tag 15-183
in XLATL tag 15-198
MSG (message) tag
description of 15-114
example of 15-115
syntax for 15-114
MSGMBR (message member) tag
description of 8-1, 15-116
example of 8-1, 15-116
syntax for 15-116
MSGTYPE attribute
in MSG tag 15-114
in PANEL tag 15-133
multiple-choice field
defining 5-15
description of 5-15
example of 5-15
multiple-choice selection field
defining 15-162
multiple-choice selection list
defining 5-28, 15-162

N

name
in CMD tag 15-35
in SELLST tag 15-161
NAME attribute
identifying variables 4-2
in CHOICE tag 15-28
in CMD tag 15-34
in DTAFLD tag 15-60
in ICON tag 15-81
in KEYL tag 15-91
in LSTCOL tag 15-101
in LSTFLD tag 15-106
in MSGMBR tag 15-116
in PANEL tag 15-132
in SELFLD 15-156
in SELLST tag 15-161
in UC tag 15-173
in VARCLASS tag 15-178
in VARDCL tag 15-186
using with the panel tag 3-1
name list
definition of X-4

national language separator 4-4
nested tags
 how to read 15-2
nesting
 in warning statement 6-29
 lists within lists 6-25
 ordered list 6-15
 parameter list 6-23
 simple list 6-12
 unordered lists 6-14
nesting tags 2-3
NOMATCH attribute 9-1
 in CMDTBL tag 15-42
NONE
 in XLATL tag 15-198
NOTE tag
 description of 6-27, 15-119
 example of 6-27, 15-120
 syntax for 15-119
note text
 in NOTE tag 15-119
 in NT tag 15-121
note (NT) tag
 description of 15-121
 example of 15-121
 syntax for 15-121
NT (note) tag
 description of 6-27, 15-121
 example of 6-28, 15-121
 syntax for 15-121
null end tags 11-6
NUMERIC attribute
 in VARCLASS tag 15-182
numeric variables
 converting 4-7
 defining 4-8
 description of 4-7
 uses of 4-7
NUMROWS attribute
 in LSTFLD tag 15-107
 in SELLST tag 15-163

O

OL (ordered list) tag
 description of 15-123
 example of 15-124
 syntax for 15-123
omitted end tags 11-3
ordered list
 defining LI tags within 15-92
 defining LP tag within 15-99
 defining within DD tag 15-44
 description of 6-15
 tag example 6-16
ordered list (OL) tag
 description of 15-123
 syntax for 15-123
ordered list (OL) tag *(continued)*
 tag example 15-124
output
 retrieving and translating 15-194
output area
 defining with data field tag 15-60
output data field
 defining 5-5
output field
 definition of X-4
overriding variable classes 4-17

P

P (paragraph) tag
 description of 6-2, 15-125
 example of 6-2, 15-125
 syntax for 15-125
PANDEF attribute
 in PANEL tag 15-132
PANDEF (panel default) tag
 description of 15-129
 example of 15-130
 using 11-1
panel
 See also dialog element
 arranging fields 15-149
 declaring variables for 4-1
 defining 3-1
 defining a region 3-21
 defining action bar and pull-downs 3-6
 defining command area 3-25
 defining common attributes and values 11-1
 defining common elements 3-1
 defining cursor placement 3-4
 defining defaults 15-129
 defining fields 5-1
 defining information region 15-82
 defining key mapping list 3-3
 defining scrollable area 3-15
 defining size 3-2
 defining the window title 3-2
 defining top and bottom instructions 3-19
 defining with PANEL tag 3-1, 15-132
 definition of X-4
 determining width and depth 3-2
 displaying 12-8
 establishing variable access 15-188
 help 1-5
 placing cursor in data field 15-60
 providing help for 3-4
 removing from library 13-10
 separating regions 3-20
panel body
 definition of X-4
 description of 1-4
panel default identifier
 in PANDEF tag 15-129

panel default identifier *(continued)*
in PANEL tag 15-132
panel default tag
using 11-1
panel default (PANDEF) tag
description 15-129
example of 15-130
panel defaults
defining 11-1
PANDEF (panel default) tag defining 15-129
panel definition
definition of X-4
panel element
See also DTL element
definition of X-4
panel ID
application panel 1-4
definition of X-4
help panel 1-6
specifying on PANEL tag 3-1, 15-132
panel name
in PANEL tag 15-132
PANEL tag
defining messages 3-3
description of 3-1, 15-132
example of 15-136
syntax for 15-132
using KEYLIST attribute 3-3
using the HELP attribute 3-4
using the NAME attribute 3-1
PANEL (panel) tag
defining cursor placement 3-4
example of 3-1
PANELID attribute
in CMDACT tag 15-38
panel, application
description of 1-4
layout 1-4
paragraph
defining within DD tag 15-44
paragraph text
in P tag 15-125
paragraph (P) tag
description of 15-125
example of 15-125
syntax for 15-125
parameter
definition of X-4
list 6-22
parameter data
in UC tag 15-174
parameter description
in PD tag 15-140
parameter description tag
description of 6-22
example of 6-22
parameter description (PD) tag
description of 15-140
parameter description (PD) tag *(continued)*
example of 15-141
syntax for 15-140
parameter entity
definition of X-4
description of 2-10
example 2-11
naming conventions 2-12
parameter list
description of 6-22
nesting tags 6-23
parameter list tag
description of 6-22
parameter list (PARML) tag
description of 15-137
example of 15-138
syntax for 15-137
parameter term
in PT tag 15-147
parameter term tag
description of 6-22
example of 6-22
parameter term (PT) tag
description of 15-147
example of 15-147
syntax for 15-147
parameters
in ACTION tag 15-7, 15-8
in CMDACT tag 15-39
PARM
in XLATL tag 15-198
PARML (parameter list) tag
description of 6-22, 15-137
example of 6-22, 15-138
syntax for 15-137
PASSTHRU
description of 9-5
in CMDACT tag 15-38
using 9-5
PD (parameter description) tag
description of 6-22, 15-140
example of 6-22, 15-141
syntax for 15-140
PDC (pull-down choice) tag
description of 15-142
example of 3-8, 15-144
syntax for 15-142
PDSEP (pull-down separator) tag
description of 15-145
example of 3-8, 15-145
syntax for 15-145
pm class name
in UC tag 15-173
in VARLIST tag 15-188
in XLATL tag 15-198
PMTLOC attribute 5-1
in CMDAREA tag 15-40
in DTAFLD tag 15-63

PMTLOC attribute *(continued)*
in SELFLD tag 15-157
PMTWIDTH attribute 5-1, 5-24
in DTACOL tag 15-56
in DTAFLD tag 15-62
in SELFLD tag 15-157
pointer, definition of X-4
pointing device, definition of X-4
pop-up window
definition of X-4
displaying messages 8-1
positioning relative to data field 15-60
positioning relative to information region 15-82
position value
in PANEL tag 15-134
predefined entities 2-12
primary window
definition of X-4
PRMDATA attribute
in UC tag 15-174
program variable
definition of X-4
prompt
See field prompt
prompt text
in DTAFLD tag 15-63
prompt-width
specifying for data field 5-4, 15-62
PT (parameter term) tag
description of 6-22, 15-147
example of 6-22, 15-147
syntax for 15-147
pull down description text
in PDC tag 15-143
pull-down
creating a horizontal divider line 3-7
defining a choice 3-6, 15-142
defining mnemonics 3-9, 5-19
pull-down choice
defining 15-142
defining action 3-9
defining current sort option 3-14
defining mnemonics 3-9
description of 3-6
example of 3-10
making it unavailable 3-12
providing help for 3-10
pull-down choice (PDC) tag
description of 15-142
example of 15-144
syntax for 15-142
pull-down separator (PDSEP) tag
description of 3-8, 15-145
example of 15-145
syntax for 15-145
pushbutton
action-type selection field 5-17, 15-157
default action 15-29
pushbutton *(continued)*
definition of X-4

R

radio button
definition of X-4
RANGE attribute
in CHECKI tag 15-22
range test 4-15
reason code
definition of X-4
reference phrase
defining in help panels 7-8
in RP tag 15-152
reference phrase help
definition of X-4
reference phrase (RP) tag
description of 15-152
example of 15-152
syntax for 15-152
REGION tag
description of 15-149
example of 15-150
syntax for 15-149
REGION (region) tag
example of 3-21, 3-24
how to use 3-21
regions
REGION tag defining 15-149
region, panel
defining 3-21
separating 3-20
relationship to CUA 1-2
removing extraneous blanks 4-15
REQUIRED attribute 5-12
in DTAFLD tag 15-61
in LSTCOL tag 15-102
RESULT attribute
in ASSIGNI tag 15-14
RETRIEVE attribute
in CMDACT tag 15-39
return code
definition of X-5
RIGHT attribute
in CMDACT tag 15-39
risk
warning statement 6-29
ROOT attribute
in ISYN tag 15-84
root word
in ISYN tag 15-84
root word list
in ITOP tag 15-86
ROOTS attribute
in ITOP tag 15-86
RP (reference phrase) tag
description of 15-152

RP (reference phrase) tag *(continued)*
example of 15-152
syntax for 15-152
RULES attribute
in LSTFLD tag 15-108
RUN attribute
example of 9-2
in ACTION tag 15-7
specifying a command 3-9, 9-2

S

SBCS
See single-byte character set
screen, definition of X-5
scroll bar
definition of X-5
horizontal 6-1
scrollable area
AREA tag defining 15-11
scrollable list
defining 15-106
defining columns of data 15-101
scrollable panel area
defining with AREA tag 3-15
description of 15-11
scrollable selection field
defining 3-16, 15-161
SELCOL (selection column) tag
description of 5-26, 15-154
example of 15-154
syntax for 15-154
selected pull-down choice 3-14
selection choice
action 15-29
availability of 5-20, 15-28
defining 15-28
defining space for 5-23
help for 15-28
multiple-choice 15-29
preselected 15-29
pushbutton 15-29
single-choice 15-29
text 15-29
selection choice tag
description of 15-28
selection choice (CHOICE) tag
description of 15-28
example of 15-31
syntax for 15-28
selection column (SELCOL) tag
description of 15-154
example of 15-154
syntax for 15-154
selection columns
SELCOL (selection column) tag defining 15-154
selection field
coding for action 15-158
selection field *(continued)*
coding unavailable choice 5-20
defining 5-13
defining a field prompt 5-1
defining an item 15-28
defining column widths for 15-56
defining cursor placement 3-4
defining immediate-action field 5-17
defining mnemonics 5-19
defining scrollable area 3-16
defining space for choice 5-23
definition of X-5
help for 5-19
pushbutton 5-13, 15-157
tag example 3-16
using the CHOICE tag 5-13
selection field (SELFLD) tag
example of 15-158
syntax for 15-156
tag description 15-156
selection fields
CHOICE tag within 15-28
M (mnemonic) tag within 15-110
SELFLD (selection field) tag defining 15-156
selection list
controlling size 5-29
defining 5-26
defining an item 15-28
defining cursor placement 3-4, 3-6
definition of X-5
determing single or multi-choice 5-27
determining single or multi-choice 15-162
help for 5-30
index 15-162
overriding the variable declaration 5-30
scalar selection variable 5-28
scrollable list 15-161
sizing 5-36
specifying preselected items 5-27
selection list (SELLST) tag
example of 15-164
syntax for 15-161
tag description 15-161
selection lists
SELCOL (selection column) tag within 15-154
SELLST (selection list) tag defining 15-161
selection width
defining 5-24
SELFLD (selection field) tag
description of 15-156
example of 15-158
syntax for 15-156
SELLST (selection list) tag
description of 5-26, 15-161
example of 15-164
syntax for 15-161
SELVAR attribute
in SELLST tag 15-162

SELWIDTH attribute 5-23, 5-24
in DTACOL tag 15-56
in SELFLD tag 15-157
SETVAR attribute 5-18
in ACTION tag 15-8
SETVERB
description of 9-5
in CMDACT tag 15-38
shift state
in KEYI tag 15-88
shift-in (SI) character
definition of X-5
shift-out (SO) character
definition of X-5
simple list
defining LI tags within 15-92
defining LP tag within 15-99
defining within DD tag 15-44
description of 6-11
tag example 6-12
simple list (SL) tag
description of 15-165
example of 15-165
syntax for 15-165
single-byte character set (SBCS)
definition of X-5
single-choice field
defining 5-14
description of 5-14
example of 5-14
single-choice selection field
defining 15-162
single-choice selection list
defining 5-28, 15-162
size, window
determining width and depth 3-2
SKIPVAR attribute
in CMD tag 15-34
SL (simple list) tag
description of 15-165
example of 15-165
syntax for 15-165
sort key text
in SORTKEY tag 15-167
sort key (SORTKEY) tag
description of 15-167
example of 15-168
syntax for 15-167
sort option
for pull-down choices 3-14
SORTKEY (sort key) tag
description of 15-167
example of 15-168
syntax for 15-167
SOURCE attribute
in ICON tag 15-81
source file
compiling 13-2
source file *(continued)*
compiling and verifying 12-2
defining dialog elements 2-1
defining frequently used words (entities) 2-6
DOCTYPE declaration 14-2
including comments 2-5
source file name
in ICON tag 15-81
specifying a list of values to match value of user input 4-17
specifying drive and path 12-7
specifying system information 12-5
specifying valid characters for data entry 4-16
stacked VDEFINE
definition of X-5
start tag delimiters 2-1
STATE attribute
in KEYI tag 15-88
static text 6-1
STDDATE attribute
in VARCLASS tag 15-181
STDTIME attribute
in VARCLASS tag 15-180
STRIP
in XLATL tag 15-198
strip translation 4-15
SUFFIX attribute
in MSG tag 15-114
synonym
defining for help index 15-84
defining with ISYN tag 15-84
synonym words
in ISYN tag 15-84
syntax conventions for Dialog Tag Language 2-1
syntax diagrams
example of 12-12
how to read 12-11
system command
ENTER and CANCEL 9-3
system extension
definition of X-5
system information
specifying 12-5
system variable
definition of X-5

T

T (truncation) tag
description of 9-7, 15-169
example of 9-7, 15-170
syntax for 15-169
tag
See also Dialog Tag Language (DTL)
attributes, description of 2-1
coding text 2-2
definition of X-5
for information region 6-1

tag *(continued)*
nesting 2-3, 6-25
values, description of 2-1
verifying syntax 12-2
tag text 2-2
tags
AB 15-3
ABC 15-5
ACTION 15-7
AREA 15-11
ASSIGNI 15-14
ASSIGNL 15-16
BOTINST 15-18
CAUTION 15-20
CHECKI 15-22
CHECKL 15-26
CHOICE 15-28
CMD tag 15-34
CMDACT 15-37
CMDAREA 15-40
CMDTBL 15-42
DATFLD 15-60
DD 15-44
DDHD 15-46
DIVIDER 15-48
DL 15-50
DT 15-54
DTACOL 15-56
DTAFLDD 15-66
DTHD 15-68
FIG 15-70
FIGCAP 15-73
HELP 15-75
Hn 15-77
ICMD 15-79
ICON 15-81
INFO 15-82
ISYN 15-84
ITOP 15-86
KEYI 15-88
KEYL 15-91
LI 15-92
LINES 15-95
LIT 15-97
LP 15-99
LSTCOL 15-101
LSTFLD 15-106
M 15-110
MSG 15-114
MSGMBR 15-116
NOTE 15-119
NT 15-121
OL 15-123
P 15-125
PANDEF 15-129
PARML 15-137
PD 15-140
PDC 15-142
tags *(continued)*
PDSEP 15-145
PT 15-147
REGION 15-149
RP 15-152
SELCOL 15-154
SELFLD 15-156
SELLST 15-161
SL 15-165
SORTKEY 15-167
T 15-169
TOPINST 15-171
UC 15-173
UL 15-176
VARCLASS 15-178
VARDCL 15-186
VARLIST 15-188
VARSUB 15-190
WARNING 15-192
XLATI 15-194
XLATL 15-197
XMP 15-202
test value
in ASSIGNI tag 15-14
text
adding to a list 6-24
defining for information region 6-2
in ABC tag 15-5
in BOTINST tag 15-18
in CAUTION tag 15-20
in CHOICE tag 15-29
in CMDAREA tag 15-40
in DTAFLD tag 15-63
in DTAFLDD tag 15-66
in DTHD tag 15-68
in FIGCAP tag 15-73
in Hn tag 15-77
in ITOP tag 15-86
in KEYI tag 15-89
in LI tag 15-92
in LINES tag 15-95
in LP tag 15-99
in MSG tag 15-114
in NOTE tag 15-119
in NT tag 15-121
in P tag 15-125
in PANEL tag 15-134
in PD tag 15-140
in PDC tag 15-143
in PT tag 15-147
in RP tag 15-152
in SELFLD tag 15-157
in SORTKEY tag 15-167
in TOPINST tag 15-171
in UC tag 15-174
in WARNING tag 15-192
in XMP tag 15-202
providing caution to 6-30

text *(continued)*
providing notes to user 6-27
providing warning to user 6-29
using the example tag 6-6
using the figure tag 6-7
using the heading tags 6-4
using the LINES tag 6-5
using the paragraph tag 6-2
text, help 7-3
text, tag 2-2, 6-1
time
in VARCLASS tag 15-179, 15-180
time and date variables
converting the format 4-4
display length 4-18
error messages 4-5
example of 4-6
national language separator 4-4
retrieving time with ZITIME 4-5
retrieving time with ZSTDTIME 4-5
types of 4-4
uses of 4-4
using IDATE 4-4
using ITIME 4-4
using STDDATE 4-4
using STDTIME 4-4
title
defining for panel 3-2
in HELP tag 15-75
title text
in PANEL tag 15-134
TOGVAR attribute 5-18
in ACTION tag 15-8
top instruction
defining 3-19
description of 1-4
top instruction (TOPINST) tag
description of 15-171
example of 15-171
syntax for 15-171
topic, help
defining name of 15-86
defining synonyms 15-84
TOPINST (top instruction) tag
description of 15-171
example of 3-19, 15-171
syntax for 15-171
TOPROW attribute 2-2, 5-33
in LSTFLD tag 15-107
in SELLST tag 15-162
total digits
in VARCLASS tag 15-182
translate item (XLATI) tag
description of 15-194
example of 15-196
syntax of 15-194
translate list
specifying individual list element 15-194
translate list *(continued)*
user exit 4-17
translate list (XLATL) tag
description of 15-197
example of 15-200
syntax for 15-197
translate lists
defining 4-10
types of 4-10
XLATI (translate item) tag within 15-194
XLATL (translate list) tag defining 15-197
translating a variable 4-10
translation
defining items for 4-9
example of 4-11, 4-12
translation parameter
in XLATL tag 15-198
truncating commands 9-7
truncation (T) tag
description of 15-169
example of 15-170
syntax for 15-169
TSIZE attribute
in DL tag 15-50
in PARML tag 15-137
tutorial
definition of X-5
TUTORIAL attribute
in HELP tag 15-75
TYPE attribute 5-8, 5-13
in CHECKI tag 15-22
in DIVIDER tag 15-48
in SELFLD tag 15-156
in SELLST tag 15-162
in VARCLASS tag 15-178
type of data
in VARCLASS tag 15-178
type of translation
in XLATL tag 15-198

U

UC (user control) tag
description of 15-173
example of 15-175
syntax for 15-173
UL (unordered list) tag
description of 15-176
example of 15-177
syntax for 15-176
unavailable choice
coding for 5-20
definition of X-5
description of 3-12, 5-20
unavailable emphasis
definition of X-5
example of 3-13
for pull-down choice 3-12

unclosed end tags 11-6
unclosed start tags 11-6
unformatted text
 using the example tag 6-6
 using the figure tag 6-7
 using the LINES tag 6-5
unordered list
 defining levels of 6-14
 defining LI tags within 15-92
 defining LP tag within 15-99
 defining within DD tag 15-44
 description of 6-13
 example of nesting tags 6-14
 tag example 6-14
unordered list (UL) tag
 example of 15-177
 syntax for 15-176
 tag description 15-176
UPPER
 in XLATL tag 15-198
upper translate list
 example of 4-9
 translating a value to uppercase 4-9
USAGE attribute 5-30
 in DTAFLD tag 15-61
 in LSTCOL tag 15-102
user control name
 in UC tag 15-173
user control (UC) tag
 description of 15-173
 example of 15-175
 syntax for 15-173
user exit
 check 4-17
 translate 4-14
 variable access 4-1

V

validating input value for variable class 15-24
validating variables 4-9
validity check item (CHECKI) tag
 description of 15-22
 example of 15-25
 syntax for 15-22
validity check list
 example of 8-4
 using 8-4
validity check list tag
 description of 8-4
 example of 8-4
validity check list (CHECKL) tag
 description of 15-26
 example of 15-27
 syntax for 15-26
validity check lists
 CHECKI (validity check item) tag within 15-22
 (CHECKL) check list tag defining 15-26
validity checks
 description of 4-15
 types of 4-15
 use of 4-15
 using CHECKL and CHECKI tags 4-15
VALUE attribute
 in ASSIGNI tag 15-14
 in XLATI tag 15-194
VALUES attribute
 in CHECKI tag 15-23
values test
 description of 4-17
 example of 4-17
value, tag
 description of 2-1
VAR attribute
 in VARSUB tag 15-190
VARCLASS attribute 5-13, 5-34
 in DTAFLD tag 15-61
 in LSTCOL tag 15-102
 in SELCOL tag 15-154
 in VARDCI tag 15-186
 in VARSUB tag 15-190
VARCLASS (variable class) tag
 attributes 4-3
 description of 15-178
 example of 15-183
 location in source file 4-3
 syntax for 15-178
 using 4-3
 using MSG attribute 8-3
VARDCL (variable declaration) tag
 attributes 4-1
 description of 15-186
 example of 15-186
 syntax for 15-186
variable class
 See also Dialog Manager Guide and Reference
 associating translate list 4-9
 defining 4-2
 defining translate list 15-197
 description 1-8
 description of 4-1
 display length 4-18
 overriding 4-17
 overriding the data variable 5-34
 purpose of 4-3
 types of 4-3
 validating input value for 15-24
 VARCLASS (variable class) tag defining 15-178
 VARLIST (variable list) tag within 15-188
 XLATL (translate list) tag within 15-197
 (CHECKL) check list tag within 15-26
variable class name
 in DTAFLD tag 15-61
 in LSTCOL tag 15-102
 in SELCOL tag 15-154
 in VARCLASS tag 15-178

variable class name *(continued)*
in VARDCL tag 15-186
in VARSUB tag 15-190
variable class table
defining information within 4-10
local variable class 4-2
variable class (VARCLASS) tag
description of 15-178
example of 15-183
syntax for 15-178
variable declaration
with VARDCL (variable declaration) tag 15-186
variable declaration (VARDCL) tag
description of 15-186
example of 15-186
syntax for 15-186
variable list
VARDCL (variable declaration) tag within 15-186
VARLIST (variable list) tag defining 15-188
variable list (VARLIST) tag
example of 15-188
syntax for 15-188
tag description 15-188
variable name
in ACTION tag 15-8
in CHOICE tag 15-28
in CMD tag 15-34, 15-35
in LSTFLD tag 15-106
in PDC tag 15-142
in SELLST tag 15-162
in VARSUB tag 15-190
variable names
assigning as a value 2-2
variable pool
saving translated input 15-194
variable substitution tag
using in a message 8-4
variable substitution (VARSUB) tag
description of 15-190
example of 15-191
syntax for 15-190
variable translate list
defining 4-9
relation to VARCLASS definition 4-9
types of 4-9
variable validation 4-9
variables
See also Dialog Manager Guide and Reference
assigning a value 15-14
character 4-3
declaring for panels, messages, and commands 4-1
defining list of checks 15-26
display length 4-18
in data fields 5-5
item translating 4-11
local variable class 4-2
numeric 4-7
variables *(continued)*
setting 5-18
specifying in message text 8-4
substituting variables in messages 15-190
supplying information about variables 15-186
time and date 4-4
toggling 5-18
translate list 4-9, 15-198
translating 4-10, 4-11, 15-194
validity checks 4-15
variable class tag 15-178
variable classes 1-8
variable declaration tag 15-186
variable list tag 15-188
variable pool 15-199
variable substitution (VARSUB) tag 15-190
VARLIST (variable list) tag
description of 15-188
example of 15-188
syntax for 15-188
using 4-1
VARSUB (variable substitution) tag
description of 15-190
example of 15-191
syntax for 15-190
verifying data input by the user
verifying tag syntax 12-2
virtual key
in KEYI tag 15-88

W

warning message
defining 8-3
definition of X-5
description of 6-29, 8-3
example of 6-29
WARNING tag
description of 6-29, 15-192
example of 6-29, 15-193
syntax for 15-192
width 3-2
width and depth
defining with PANDEF tag 11-1
determining 3-2
WIDTH attribute
in FIG tag 15-70
in INFO tag 15-82
in PANDEF tag 15-129
in PANEL tag 15-133
window
See also panel
defining width and depth 3-2
definition of X-5
layout 1-4
resizing 3-15
title 3-2

window size
 defining with PANEL tag 3-2
 determining width and depth 3-2
window title
 defining with PANEL tag 3-2
 definition of X-5
 description of 1-4
 for help panels 1-6
word-wrapping 5-7, 5-10

X

XLATI (translate item) tag
 description of 15-194
 example of 15-196
 syntax of 15-194
XLATL (translate list) tag
 description of 15-197
 example of 15-200
 syntax for 15-197
 using MSG attribute 8-4
XMP (example) tag
 description of 6-6, 15-202
 example of 6-7, 15-203
 syntax for 15-202

Z

ZCMD system variable
 for PASSTHRU 9-5
 for SETVERB 9-5
ZVERB system variable
 for SETVERB 9-5

Special Characters

% notation 2-2
 definition of X-5

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark by Rosendahl · Esbjerg

